For Macintosh Programmers & Developers

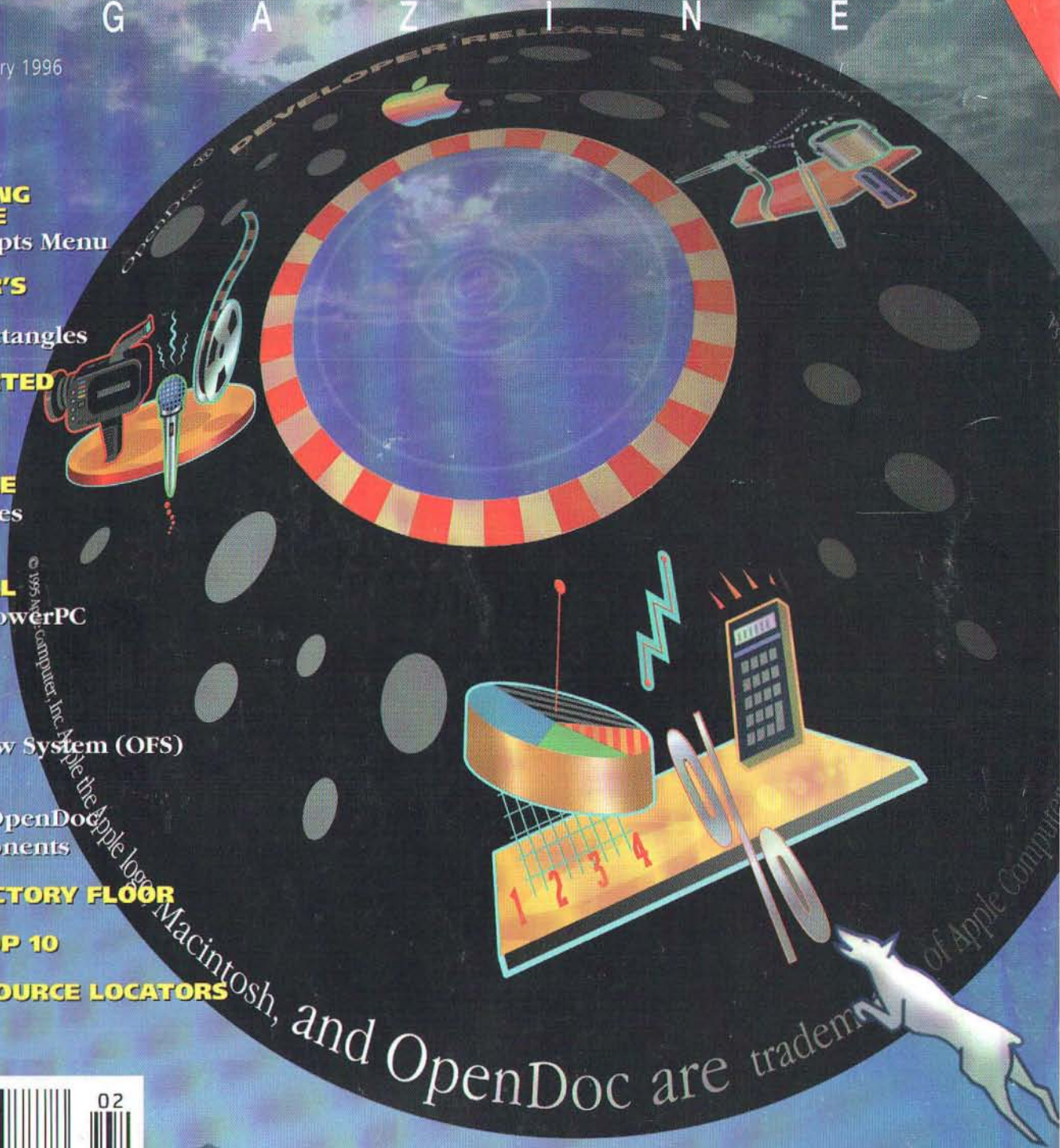
MacTech™ MAGAZINE

ol. 12, No. 2 • February 1996

In This Issue!

OpenDoc DR4 SDK CD with Cyberdog!

INSIDE:



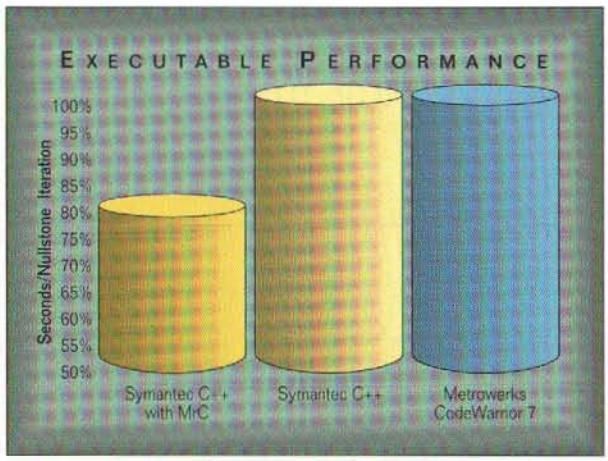
0 32128 74887 8 02

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Now, through a joint development agreement, Symantec and Apple™ Computer let you produce the fastest Power Mac code.

Symantec C++ for Power Macintosh™ now comes with MrC– Apple's new optimizing compiler.

Industry-standard Nullstone tests show that



*Code compiled using MrC runs an average of 22% faster than code compiled with the standard Symantec C++ compiler or Metrowerks Code Warrior 7™.**

size and color. Organizing and navigating a project has never been easier.

In addition, Symantec C++'s multi-threaded environment gives you the ability to edit and write code while you compile. And our visual architect lets you quickly draw the interface. Corresponding code is then generated automatically.

WHEN MRC GOES TO WORK, SYMANTEC C++ APPLICATIONS GET 22% FASTER.

applications compiled with MrC run an average of 22% faster.

DEVELOP FASTER APPLICATIONS FASTER.

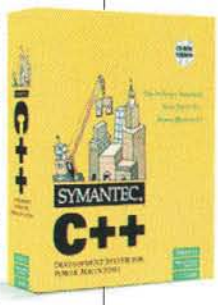
Not only can you develop the fastest Power Mac applications, you can write them fast, too. New AppleScript support lets you automate repetitive tasks. While the new linker provides fast turnaround for incremental builds.

With the fully integrated class browser, you'll quickly navigate your C++ class library. And support for templates and multiple inheritance further boosts your productivity.

ORGANIZE AND MANAGE PROJECTS EASILY.

The new Project Manager lets you organize and manage nested projects. You can display hierarchical groups as folders within a project window for easier organization.

The editor gives you unlimited split panes and full text formatting– font,



MRC COMPILER
produces the fastest Power Mac applications

APPLESCRIPT
automates the build process

NEW INCREMENTAL LINKER
provides fast incremental builds

TEMPLATE AND MULTIPLE INHERITANCE SUPPORT
increases productivity

NESTED PROJECTS AND FOLDERS
let you organize and navigate projects

MULTI-THREADED ENVIRONMENT
lets you edit and write code while compiling

VISUAL ARCHITECT
builds your interface visually

All in all, Symantec C++ is a great way to develop the fastest applications for Power Mac.

THREE CDS FOR THE PRICE OF ONE.

When you register as a Symantec C++ owner, you'll be enrolled in the Symantec C++ Subscription for Macintosh program. Subscribers will automatically receive two free product updates (on CD ROM) so you'll always have the latest features and tools.

Learn more about Symantec C++ on the Internet at www.symantec.com Or call 1-800-628-4777, Extension 91122 for more information.

SYMANTEC.®

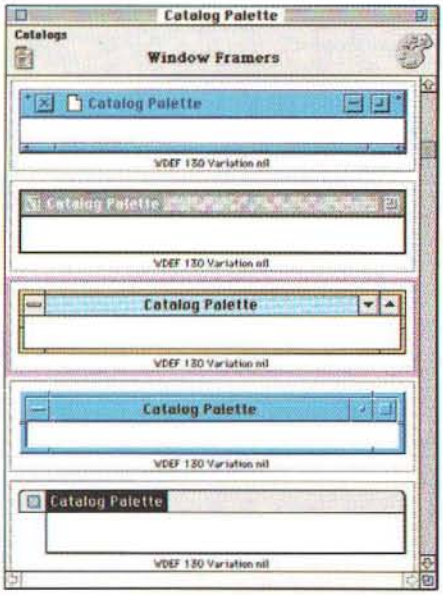
Building Internet And Other Native PowerPC Applications Has Never Been Easier Or Faster.

SmalltalkAgents®

Agents Object System
MACINTOSH

SmalltalkAgents (STA) is a sophisticated rapid application development environment based on a new generation of the Smalltalk language, enabling you to easily deliver double-clickable applications.

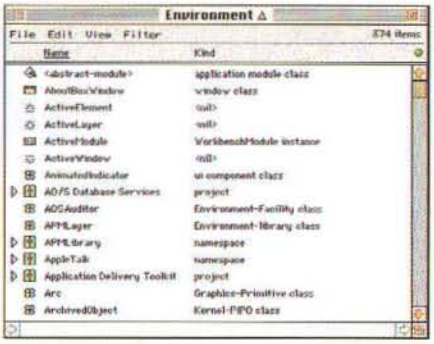
Copland Style GUI Look & Feel



Creating professional quality user interfaces is easy with our component parts libraries.

VisualWorkbench

Visually manipulate all objects including source and design elements using your mouse and keyboard. Visually manage



design and project elements in a "Finder-like" desktop workspace as fluidly as you work with folders and documents on your desktop. Interactively build, wire, and interconnect reusable components and interfaces in an integrated environment.

GUI Design & Generation

Live "Drag and Drop" manipulation to build your application's visual interface using components that "know" how to behave and autoconfigure themselves into an environment. Create new components and/or wire together existing components that can

be saved as reusable template designs for use in other applications or containers.

DTP Engine & Word Processor

STA not only includes a programmable word processor component and HyperMedia engine, but also a powerful report writer that supports embedding of any kind of objects, movies, flows, and international text, and page layout.

C/C++, Pascal Workbench

Compile, edit, and dynamically link C/C++, Pascal, Fortran, and Assembly code directly from within our STA VisualWorkbench as an integrated part of the Smalltalk application development process.

Component-based Architecture

STA components are designed for OpenDoc and OLE, and will give you transparent integration with OpenDoc and OLE when they become available.

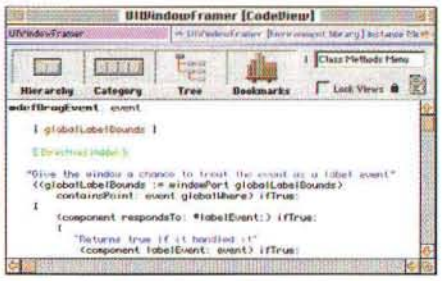
Threading & Internet Tools

STA provides powerful support for Internet

server as well as client tool development. Pre-emptive threading, thread safe libraries and classes for TCP/IP protocols are standard features enabling you to quickly and easily deliver custom e-mail, WWW, list-server, and other dial-up/network related apps.

PowerPC Support

STA provides binary portability across differ-



ent CPUs and Operating Systems. Design applications today on one platform and simply deploy on other platforms as required.

Contact us about our Web Server & Client Toolkit at 1-800-296-1339 or at <info@qks.com> or visit our Web site http://www.qks.com/.

Q·K·S Quasar Knowledge Systems, Inc.
9818 Parkwood Drive
Bethesda, MD 20814 USA
Tel:(301)530-4853 Fax:(301)530-5712

THE STAFF

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?** If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

DEPARTMENTS	Internet	CompuServe	AppleLink
Orders, Circulation, & Customer Service	custservice@mactech.com	71333,1063	MT.CUSTSVC
Press Releases	pressreleases@mactech.com	–	–
Ad Sales	adsales@mactech.com	71552,172	MT.ADSALES
Editorial	editorial@mactech.com	71333,1065	MT.EDITORIAL
Programmer's Challenge	progchallenge@mactech.com	71552,174	MT.PROGCHAL
Online Support	online@mactech.com	–	–
Accounting	accounting@mactech.com	–	–
Marketing	marketing@mactech.com	–	–
General	info@mactech.com	71333,1064	MACTECHMAG
Online support area	http://www.mactech.com	*type* GO MACTECHMAG	*see* Third Parties: Third Parties (H-O)

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Second Class postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

BULK RATE
U.S. POSTAGE
PAID
PERMIT NO. 135
MIDLAND, MI 48640

THIRD ENCLOSED

TABLE OF CONTENTS

February 1996 • Volume 12, No.2

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

PUBLISHER'S PAGE

By Neil Ticktin, Publisher

[Scott's taking a much needed break this month from his Viewpoint column – he'll be back next month with more insight and comments on our industry.]

The Psyche of "Real" Computer Users

I recently attended Comdex, and now, besides wondering *why* I took the time to go to the show, I find myself making some interesting observations about the computer industry – and more importantly, the psyche of computer users.

Late this past summer, the Fed's favorite target of anti-trust investigations released Win95. And with all the hype, even the most stout of heart in the Macintosh industry had to wonder: "Can the Macintosh survive?"

I don't think that most of us believed that Win95 could kill the Mac OS (especially with Win95 sales projections dropping nearly 50%), but a lot of us were looking for the deeper reasons as to why we felt that way. For over a decade, Mac users have have had an easy answer to this question – other systems simply lacked anything resembling a real user interface. And even though, as the bumper sticker says, "Win95 = Mac '89", many people still believe the *perception* that Microsoft wants them to – that Win95 is as good as a Mac. And while the Mac OS is still superior in many ways, it has become tougher to explain to Windows users why we use Macs – and why we won't "just convert to using Windows."

As Win95 shipped, I recall actually hypothesizing: "What would it be like if the Mac and Apple ceased to exist, and I was forced into using Windows?" My conclusion was that I would likely reduce the number of things that I used a computer for – I would turn to PDAs whenever I could. Why? Simply because I don't enjoy using Windows and I would want to minimize my exposure to it. Would you want to spend more or less time with a "friend" you didn't like?

I also concluded that FDR's "the only thing we have to fear is fear itself" is quite applicable at this time to the Mac side of our industry.

Back to my Comdex thoughts...

Walking around Macworld, I see lots of animated faces – people are excited about cool technologies, and are even irritated when they don't see enough cool things. And we know why – Macintosh users have bonded with their machines. It's more than seeing neat things done on the computer – they've integrated the machine so completely into their thoughts, that their Macs have become partners, not just tools.

PC users, in general, are different. While there are a great many exceptions, the "bond quotient" per computer is far higher on the Mac side of the industry. As I looked around Comdex, the faces of all the people were so serious looking. They appeared bored and overwhelmed. I don't remember seeing a single excited face (except for when they were watching the mimes at the front door).

I also took note that Comdex was a lot less about technology and what computers could do for you, and a lot more about marketing, flash, and sales tactics. Sometimes, we forget how different the Macintosh and Windows worlds are. For example, we know of one company that used their highly successful Windows product marketing approach in the Macintosh market – only to have it fail so miserably, it may have been better to do nothing. Food for thought.

But what does this mean, Neil?

The bottom line is this. In general, Windows users don't view their computers in the same way a Mac user views a Macintosh. Mac users have so bonded to their machines that they almost go out of their way to find new ways to use the computer. Mac users *like* their machines – Windows users think of their computer as a box they use. And until Microsoft figures how to create this kind of relationship between PC and user, the Macintosh is still the superior user interface and product. By the way: I don't expect that Windows users will "get" this argument – if they did, they'd use the Macintosh.

Save Cyberdog!

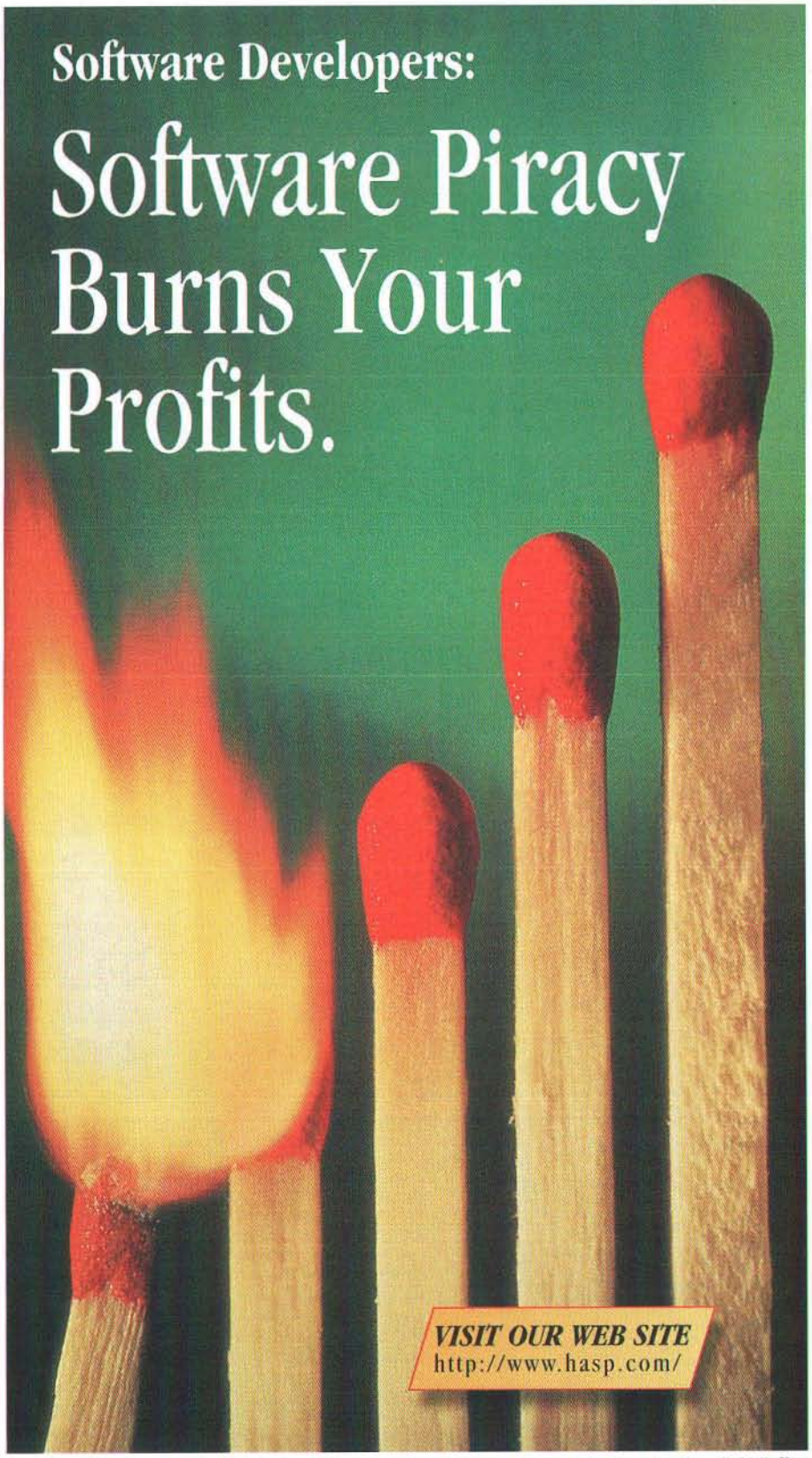
Apple has been working on a set of Internet tools based on OpenDoc. While this product will probably end up named something like "OpenDoc Internet Client", its code/commonly known name is Cyberdog, which comes from a *New Yorker* cartoon showing two dogs at a terminal, where one dog looks at the other and says "On the Internet, no one knows you are a dog."

Unfortunately, there seem to be some trademark issues for the use of the Cyberdog name. It also may be the case that Cyberdog isn't as "professional" a name as the corporate types may want. But, do Mac users want professional? or do Mac users want cool, fun computers?

Want to do your part to save the "Cyberdog" name? You can. Send e-mail to save-cyberdog@mactech.com. We're going to compile the comments and pass them onto Apple, so even if your message is short and sweet, your vote will be counted.

Software Developers:

Software Piracy Burns Your Profits.



Each year, the illegal use of software consumes nearly 50% of your potential revenues. With the flames of piracy eating away at your profits, can you afford not to protect your software?

Software Obtained Illegally, by region, 1993 vs. 1994

MacHASP® is widely acclaimed as the world's most advanced software protection solution for Macintosh computers. Since 1984, thousands of leading Mac and PC developers have used over one million MacHASP and HASP keys to protect billions of dollars worth of software. Why? Because MacHASP's security, reliability, and ease-of-use led them to a simple conclusion: MacHASP is the most effective software protection system available.

Today, more developers are choosing MacHASP than any other software protection method. To learn why, and to see how easily you can increase your revenues, call now to order your MacHASP Developer's Kit.

1-800-223-4277

ALADDIN

The Professional's Choice

North America — **Aladdin Software Security Inc.**
Tel: (800) 223 4277, 212-564 5678
Fax: 212-564 3377
E-mail: sales@hasp.com
WWW: http://www.hasp.com/

Intl Office — **Aladdin Knowledge Systems Ltd.**
Tel: 972-3-537 5795, Fax: 972-3-537 5796
E-mail: aladdin@aladdin.co.il

United Kingdom — **Aladdin Knowledge Systems UK Ltd.**
Tel: 01753-622266, Fax: 01753-622262

France — **Aladdin France SA**
Tel: 1 40 85 98 85, Fax: 1 41 21 90 56

member of SPA

yes It runs with NetWare (Developer tested only)

Available and Ready for OS/2

MICROSOFT WINDOWS COMPATIBLE

Mac OS

■ **Aladdin Benelux** 08894 19777 ■ **Aladdin Japan** 0426 60 7191 ■ **Aladdin Russia** 095 9230588 ■ **Australia** Conlab 3 8985685 ■ **Czech** Atlas 2 786085 ■ **Chile** Micrologica 2 222 1388 ■ **Denmark** Berendson 39 577300 ■ **Egypt** Zeineldin 2 3604632 ■ **Finland** ID Systems 0 870 3520 ■ **Germany** CSS 201 278804 ■ **Greece** Unibrain 1 6896320 ■ **India** Solutions 11 2218254 ■ **Italy** Partner Data 2 2614/380 ■ **Korea** Dae A 2 848 4481 ■ **Mexico** SSoft 5 5439770 ■ **New Zealand** Training 4 5666014 ■ **Poland** Systherm 61 480273 ■ **Portugal** Futurmatica 1 4116269 ■ **Romania** Interactiv 64 153112 ■ **South Africa** D Le Roux 11 886 4704 ■ **Spain** PC Hardware 3 4493193 ■ **Switzerland** Opag 61 7169222 ■ **Taiwan** Teco 2 555 9676 ■ **Turkey** Mikrobeta 312 467 7504

Power Too Abundant to Meter

neoAccess™
Cross-Platform Object Database Engine

Powerful
NeoAccess and NeoShare are the most powerful object-oriented database engines available. They display electrifying performance—up to ten times that of competitors. Behind an elegant programming interface is a high performance query engine utilizing: extended binary trees and binary search algorithms tuned for short access times, dynamically combined, collapsed, and compressed indices, and object caching for lightning fast access to previously used objects.

No Runtime Fees
Get the power of NeoAccess and NeoShare, and avoid the expense and administrative hassle of feeding the runtime fees meter. You pay one affordable price no matter how many copies of your application you sell or use.

Cross Platform
Others may promise cross-platform development tools—NeoLogic delivers. NeoAccess is a set of C++ classes designed for use with popular compilers and application frameworks on Windows®, Macintosh®, and Unix™ platforms. Full source code is included so it can even be used with custom frameworks.

New in NeoAccess Version 4.0
NeoAccess now supports Microsoft Visual C++ 4.0 IDE, MFC 4.0 and the latest version of Metrowerks CodeWarrior. A new stored query class increases speed and flexibility in accessing objects. Inverted indices and full word search capabilities ease working with many-to-many relationships. And all new on-line Hyper-Reference guide, documentation, tutorial, and sample applications get you up to speed quickly and provide a complete set of reference material.

neoShare™
Client/Server Object Database Engine

Scalable
NeoShare extends the core features of NeoAccess to client/server applications. NeoShare includes the NeoAccess Toolkit and everything you need to create data intensive distributed applications. Its advanced client/server architecture provides shared access to objects by multiple networked clients. Combine client and server functions into a single application or create separate client and server applications. With NeoShare you have complete flexibility in the design of your distributed systems.

Proven
Thousands of commercial and in-house developers have already found that NeoLogic's technology enabled them to build fast, powerful applications in record time. That's why NeoAccess and NeoShare based applications are already operating on millions of computers. Tap into the power for your next development project!

neo•logic™

Powering Development of Object-Oriented Applications

NeoLogic Systems 1450 Fourth St., Suite 12 v. 510.524.5897
neologic@neologic.com Berkeley, CA 94710 f. 510.524.4501

GETTING STARTED

By Dave Mark, MacTech Magazine Regular Contributing Author

PowerPlant and Commanders

This month, we're going to explore a brand new aspect of PowerPlant: the concept of **commands, commanders,** and the `LCommander` PowerPlant class. PowerPlant commands are similar to messages. You've already seen how messages are sent from a broadcaster (such as a button) to all listeners registered to listen to that broadcaster. The command model is slightly different.

Commands are associated with menus and keyDown events. To respond to menu selections and user keystrokes, you'll need to create a class derived (at least in part) from the `LCommander` class. There are three key functions you'll inherit and override from the `LCommander` base class.

- `HandleKeyPress()` – receives an EventRecord containing a character typed by the user.
- `ObeyCommand()` – receives a command number associated with a specific menu command. When we create this month's project, you'll see how to associate a command number with a specific menu item.
- `FindCommandStatus()` – gives your `LCommander` subclass a chance to update (i.e. enable, disable, check, uncheck, change item text) the status of the menu item associated with a specified command.

Basically, PowerPlant handles the administrative work of keeping track of which menu items need to be enabled, which pane should receive which event, etc. Every PowerPlant application has a **chain of command**. The chain (really a tree) starts with the `LApplication` object and flows downward through other objects that handle commands to the panes that will become the **targets** of the commands. Think of the current target as the application's current focus. If a keystroke is entered, the corresponding keyDown event will be sent to the current target pane (perhaps a window, perhaps a textEdit pane within the window).

As you'll see in this month's program, you'll have a little setup work to do, and then you'll override the three `LCommander` functions described above. That's pretty much it. Of course, as you get deeper into PowerPlant you'll discover that there is much more you *can* do, but for now, concentrate on understanding the basics.

A Sneak Preview of BeepCommander

This month's program is called **BeepCommander**. It features a single window type that responds to keyDowns by displaying the typed character in the window. Figure 1 shows a BeepCommander window.

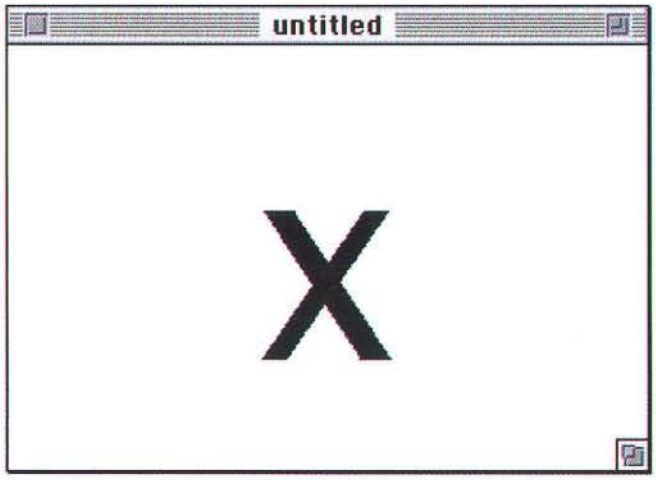


Figure 1. A BeepCommander window.

BeepCommander also features a menu named **Special** with a single item named **Beep**. When you select **Beep**, your computer beeps (gasp!). The sneaky thing is, the **Beep** item is

only enabled when the letter 'x' is typed. I know, I know, weird user interface. That's fine. The point is to show the relationship between menus, keystrokes and the LPane and LCommander functions you'll be overriding.

Let's get started.

Create a New Project

The first thing you'll need to do is create a new project, based on the PowerPlant stationery.

- Create a new folder called BeepCommander.
- Launch CodeWarrior and create a new project named BeepCommander.μ.
- In the project window, double-click on the file <PP Starter Resource>.rsrc. This will open the file in Constructor.
- In Constructor, do a **Save As...** and save the file in the BeepCommander folder as BeepCommander.rsrc.

(This last step tells Constructor to completely duplicate the file, and not just the resources it uses. This is definitely the right way to replace the stationery resource file.)

- Quit Constructor and return to CodeWarrior.
- Add the file BeepCommander.rsrc to the project.
- Delete the file <PP Starter Resource>.rsrc from the project.
- In the project window, double-click on the file <PP Starter Source>.cp.
- Select **Save As...** from the **File** menu and save the file as BeepCommander.cp.

Notice that the stationery file name changed from <PP Starter Resource>.rsrc to BeepCommander.cp in the project window. You might want to dog-ear this page and refer back to it the first few times you create your own PowerPlant projects. These first nine steps make a good starting point for all your new PowerPlant stationery-based projects.

Creating the Project Resources

Your next task is to add a new menu to the project and associate a command number with the menu's item.

- Launch your favorite resource editor.

Be sure you install the appropriate resource editing templates for your resource editor. You'll find the files PowerPlant Resorcerer TMPLs and PowerPlant ResEdit TMPLs buried in subfolders within the Metrowerks CodeWarrior folder. To install the Resorcerer templates, drag the file PowerPlant Resorcerer TMPLs into the folder Resorcerer® Templates. To install the ResEdit templates, duplicate ResEdit, then use ResEdit to edit the copy. Open the file PowerPlant ResEdit TMPLs and copy the 'TMPL' resources into your copy of ResEdit. Your copy now has the templates installed. As always, keep your original around in case things get screwed up.

- Select **Open...** from the **File** menu and open the file BeepCommander.rsrc.
- Create a new 'MENU' resource. Change its id to **131** (be sure it gets changed in both places if you are using ResEdit). Give the new menu a title of **Special** and create a single item called **Beep**. If you like, give **Beep** a command-key equivalent of **⌘B**.
- Create a 'Mcmd' resource, also with an id of **131**. Add a single item with a command number of **1000**.

The 'Mcmd' resource you just created associates a command number of 1000 with the **Special** menu's **Beep** item.
Figure 2 shows the hex version of this resource in ResEdit in case you can't get your 'Mcmd' resource template working or if you just want to check your handiwork.

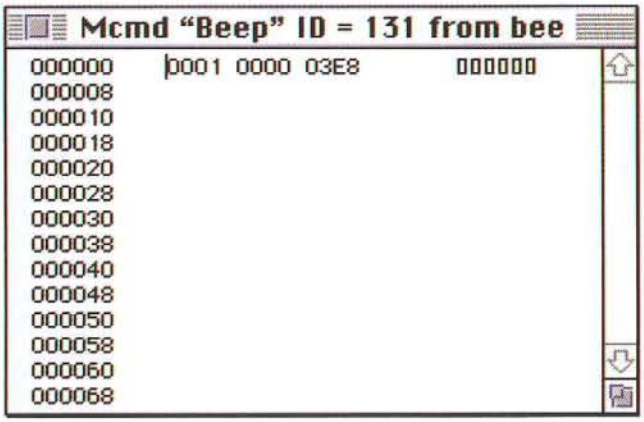


Figure 2. The hex version of 'Mcmd' 131.

- Modify 'MBAR' 128, adding the new 'MENU' id (131) to the list of other 'MENU' ids already in the resource.
- Save your changes and quit your resource editor.

Constructor

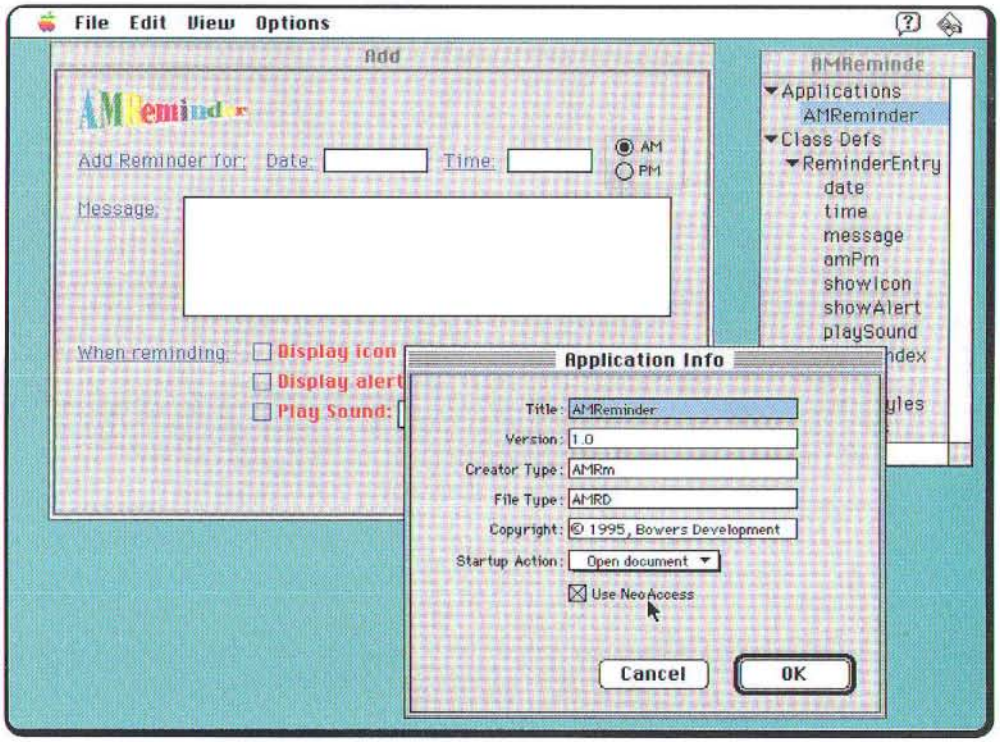
Now we'll use Constructor to create the views we'll use in this program.

- Back in CodeWarrior, double-click on the file BeepCommander.rsrc to open the file in Constructor.
- In Constructor, delete the existing "<Replace Me>" view.
- Select **New Resource** from the **Edit** menu to create a new view.
- When the New Resource dialog appears, type **Single Char Window** in the textEdit field, be sure the popup menu is set to **LWindow**, then click **OK**.
- Close the new view window.
- Be sure the new view is highlighted in the master view list, then select **Resource Info** from the **Edit** menu.

AppMaker

Your Assistant Programmer

AppMaker makes it fast and easy to build a user interface. Just point and click, then let AppMaker create resources and generate source code. AppMaker generates excellent source code for most popular languages and frameworks.



Extremely customizable and extensible—The source code generator, the resource generator, and even AppMaker's own user interface are all soft-coded so that you can easily tailor AppMaker to meet your needs.

Designed for portability—A single AppMaker document can now generate code and resources for a wide variety of languages, frameworks, and eventually, platforms.

Supports CodeWarrior PowerPlant, NeoAccess, and PowerMac development.

New features include Drag&Drop, outline view, simulator, and support for color.

Includes one-year automatic update service with at least three CDs per year.

B • O • W • E • R • S
Development

97 Lowell Road, Concord, MA 01742 • (508) 369-8175 • FAX 369-8224
CIS: 70731,3710 • AOL: BowersDev • AppleLink: D1721

- When the view info window appears, change the resource id to **1000**.
- Close the view info window.
- Double-click on the view name in the master view list to reopen the view editing window.

This view represents our main window, the window that will be created when you select **New** from our application's **File** menu. We're now going to add a pane to the window that will be reflected in our source code by the class `CSingleCharPane`. Just as a heads up, `CSingleCharPane` will be partially derived from the `LCommander` class and will be the target for both menu selections from our new `'MENU'` and for keystrokes. More on all this later.

- Drag an LPane from the palette into the center of the view editing window.
- Double-click on the LPane to open a pane info window.
- Set the Location coordinates according to those shown in Figure 3.
- Check all four of the **Binding to Superview** checkboxes, keeping the pane proportional to its enclosing window.
- Change the Pane ID to **2000**.
- Change the Class ID to **Cmdr**.

This last step is extremely important. The Class ID is what ties this view resource to the `CSingleCharPane` class we'll define when we get to the source code. By the way, just as Apple reserves all lower case resource types, Metrowerks reserves all lower case Class IDs (for example, `'abcd'` is reserved, but `'Abcd'` is just fine).

- Save your changes and quit Constructor.

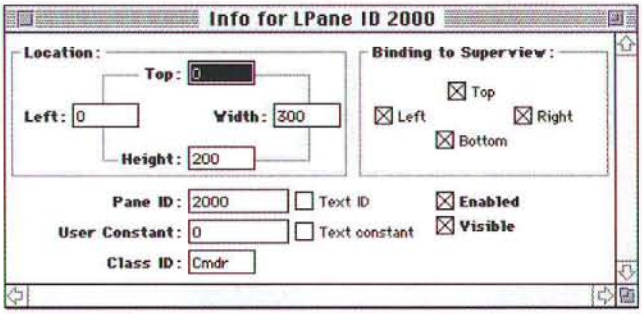


Figure 3. The pane info window for our LPane.

Adding the Source Code

Your next step is to return to CodeWarrior and type in some new source code.

- Back in CodeWarrior, create a new source code file, save it under the name `CSingleCharPane.cp`.
- Type in the following source code:

```
#include <LPane.h>
#include <LCommander.h>
#include "CSingleCharPane.h"

CSingleCharPane *
CSingleCharPane::CreateSingleCharPaneStream(
  LStream *inStream )
{
  return( new CSingleCharPane( inStream ) );
}


CSingleCharPane::CSingleCharPane( LStream *inStream ) :
  LPane( inStream )
{
  mChar = 'x';
}


Boolean
CSingleCharPane::HandleKeyPress(
  const EventRecord &inKeyEvent )
{
  mChar = inKeyEvent.message & charCodeMask;

  SetUpdateCommandStatus( true );
  Refresh();

  return true;
}


Boolean
CSingleCharPane::ObeyCommand( CommandT inCommand,
                    void *ioParam )
{
```

Announcing

MICROGUARD PLUS.™

Why so many developers are switching to MicroGuard copy protection

MicroGuard is committed to uncompromising technological superiority

"Technology at its peak" is our commitment to you. That is why we have brought you **MicroGuard Plus**™. And, MicroGuard Plus is **100% backwards** compatible with MicroGuard. This means MicroGuard and MicroGuard Plus can be used interchangeably. Just as we promised!

MicroGuard has now surpassed its own technological lead, actually improving on the best

MicroGuard Plus is everything MicroGuard is, plus 40-bit encryption, two additional passwords, 64-Bit Array, 32-byte public area, 45% size reduction, enhanced counter and more. In addition, MicroGuard Plus offers two new utilities: **QuickGuard**™ and EasyGuard™ allow you to protect your applications without touching your source code. The only feature that is not plus is the price. :-)

MicroGuard offers you the most sophisticated network protection

Our network protection, **MicroGuard Net**™, is so superior, we had to hire an Apple network engineer to execute our specifications.

MicroGuard is the best selling Macintosh key in the world

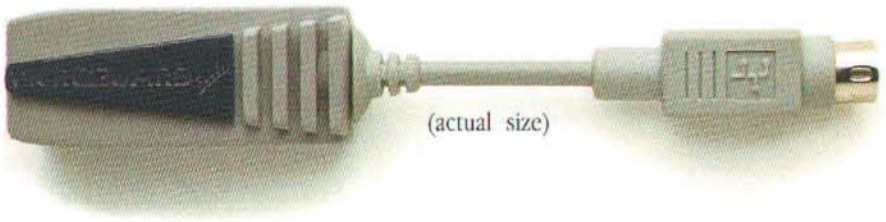
MicroGuard sells more Macintosh copy-protection keys than anyone else in the world!

MicroGuard is the only key developed by Mac developers, and is the first and only 100% ADB savvy key

We have been developing Mac applications as a seed development house since 1984. We are not a PC protection company that has come to you with a Mac product. MicroGuard is fully ADB savvy and offers extended addressing. Unlike other protection devices, MicroGuard never clashes with other keys. Only MicroGuard offers this level of sophistication.

MicroGuard delivers developer support within 24 hours

We will answer any inquiry you have within 24-hours. We also have a fully loaded AppleLink bulletin board which contains all our libraries, tech notes, Q&A and nearly everything you'll ever need!



For more information and to order a Developer's Kit or to receive a free CD ROM about MicroGuard, please contact us at:

MicroGuard USA: Tel: (303) 320-1628 • Fax: (303) 320-1599 • AppleLink: M.GUARD
International: Tel: (972) 3 558-2345 • Fax: (972) 3 558-2344 • AppleLink: MICROGUARD

Mac OS

Pirate Busters

```
  if ( inCommand == 1000 )
  {
    SysBeep( 20 );
    return true;
  }
  else
    return LCommander::ObeyCommand(inCommand, ioParam);
}


void
CSingleCharPane::FindCommandStatus(
                    CommandT inCommand,
                    Boolean  &outEnabled,
                    Boolean  &outUsesMark,
                    Char16   &outMark,
                    Str255   outName )
{
  if (inCommand == 1000)
    outEnabled = (mChar == 'x');
  else
    LCommander::FindCommandStatus(inCommand, outEnabled,
                    outUsesMark, outMark, outName);
}


void
CSingleCharPane::DrawSelf()
{
  Rect          frameRect;
  short         x, y, frameWidth, frameHeight;
  const shortk FontSize = 128;
  FontInfo      myFontInfo;

  CalcLocalFrameRect( frameRect );

  frameWidth = frameRect.right - frameRect.left;
  frameHeight = frameRect.bottom - frameRect.top;

  TextSize( kFontSize );

  x = (frameWidth - CharWidth( mChar )) / 2
     + frameRect.left;

  GetFontInfo( &myFontInfo );
  y = frameRect.bottom - ((frameHeight -
    myFontInfo.ascent + myFontInfo.descent) / 2);

  MoveTo( x, y );
  DrawChar( mChar );
}
```

Save your work, and add the file to the project.

Comments on SingleCharPane.cp

`SingleCharPane.cp` starts off with a creation function. We'll pass that in when we register this new class by calling `URegistrar::RegisterClass()`. Notice that the creation routine actually creates the `CSingleCharPane` object. Get used to this way of doing things in PowerPlant.

Next comes the constructor. Notice that the constructor maps the input parameter to the `LPane` constructor. `CSingleCharPane` is derived from both `LPane` and `LCommander`. The data member `mChar` holds the last character typed. We initialize it to 'x', since that's the magic character that enables the **Beep** item.

The function `CSingleCharPane::HandleKeyPress()` is inherited from the `LCommander` class and gets called in response to a keyDown event. The function returns `true` if the keystroke was handled correctly (in our case, we always return `true`). `LCommander::SetUpdateCommandStatus(true)` marks the menu bar as needing its status updated. `LPane::Refresh()` forces an update on the visible portion of the pane.

`CSingleCharPane::ObeyCommand()` is also inherited from `LCommander` and returns `true` if the command was obeyed. If we get command 1000 (that's the command number of the **Beep** item), we'll beep once and return `true`. Any other command causes a call to the inherited `ObeyCommand()`. This passes the command back up the chain to our commander. The `LApplication` class is the ultimate commander and has no supercommander. If the `LApplication` class can't handle your command, you are out of luck!

`CSingleCharPane::FindCommandStatus()` is inherited from `LCommander`. It checks to see if the command sent to it is 1000 (the **Beep** item). If so, it sets the `enable` parameter depending on whether `mChar` is set to 'x'. We could also have put a mark next to the **Beep** item or changed its name (try messing with these two: make the item name change to **Beep** followed by the current letter in the window, or add a checkmark next to the item when you type an 'x'). If the command wasn't a 1000, we'll pass it back up the chain.

`DrawSelf()` is an `LPane` member function. `DrawSelf()` is paired with a member function named `Draw()`. `Draw()` gets called to set up the pane's drawing environment in preparation for drawing (sort of like a call to `SetPort()`) and `DrawSelf()` is called to do the actual drawing. You might call an inherited `Draw()` method to prepare your derived pane for drawing, but you'll override the `DrawSelf()` method to provide your own drawing method.

`CSingleCharPane()` calls `CalcLocalFrameRect()` to get our pane's Rect. We'll then set the font size to `kFontSize`, do some font calculations and draw the character in the window.

By the way, if you are trying to figure out the calling sequence for an overriding function, check out the function you are overriding. For example, when you are creating `CSingleCharPane::ObeyCommand()`, check out `LCommander::ObeyCommand()` or, even better, `CPPStarterApp::ObeyCommand()`. Also, get yourself a copy of *Inside PowerPlant*, which comes on your CodeWarrior CD and contains complete descriptions of all of these routines. You can also buy a printed copy of *Inside PowerPlant* directly from Metrowerks.

Adding the Include File CSingleCharPane.h

Next, we'll create the include file `CSingleCharPane.h` that defines the `CSingleCharPane` class.

- Create a second source code file and save it as `CSingleCharPane.h`.
- Type in this source code:

```
#include <LPane.h>
#include <LCommander.h>


class  CSingleCharPane : public LPane,
  public LCommander {
```

Eddy Award Winner** for **Best New Developer Tool
– MacUser Editors Choice Awards, 1993

"A distinct improvement over ResEdit."
– MacTech / MacTutor

"Resorcerer's data template system is amazing!"
– Bill Goodman, author of Compact Pro

"Nuke ResEdit! Resorcerer is mission-critical for us."
– Dave Winer, Userland Frontier

"The color pixel editors are wonderful! A work of art!"
– Dave Winzler, author of Microseeds Redux

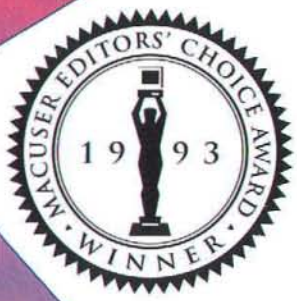
"Every Macintosh developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

"...a wealth of time-saving tools."
MacUser Review, Dec. 1992



RESORCERER®

Version 1.2.4

The Resource Editor for the Macintosh Wizard

ORDERING INFO

Needs: ≥Mac Plus, ≥ Sys 4.2, 1MB
Likes: ≥Mac Plus, ≥ Sys 7.0, 2MB
32-bit clean, AU/X compatible

Price: $256 (decimal)
(Educational, quantity, or other discounts available)

Includes: 500 page manual
60-day Money-Back Guarantee
Domestic UPS ground shipping

Payment: Check, PO's, or Visa/MC

Extras (call us):
COD, FedEx, UPS Blue/Red, International Shipping

Downloadable Demos/Updaters:
AppleLink: Software Sampler
AOL: Software Libs/Development
CompuServe: MACDEV/Tools
or call us.

New 1.2 Features:

- **New 'cicn', 'ppat', 'crsr', 'acur', 'pltt', 'clut' editors**
- **Powerful icon family editing (all 9 icon types)**
- **Color pixel anti-aliasing, dithering, and lots more**
- **Complete 'PICT' disassembly and reassembly**
- **Resource sorting; ROM resource browsing**
- **120 template field parsing types now supported**
- **New insertion & deletion template field types**
- **Text-only 'PICT' resources**
- **Lots of improvements throughout**

- Easier, faster, more Mac-like, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Help, AppleEvent, OCE, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Well-designed, helpful developer tools being added all the time
- Relied on by thousands of Macintosh developers around the world

MATHEMÆSTHETICS, INC.
P.O. Box 298 • Boulder • CO • 80306-0298 • USA
Phone: (303) 440-0707 • Fax: (303) 440-0504
AppleLink/AmericaOnline: RESORCERER • Internet: resorcerer@aol.com

```
public:
  enum { class_ID = 'Cmdr' };

  static CSingleCharPane *CreateSingleCharPaneStream(
    LStream *inStream );

  CSingleCharPane( LStream *inStream );
  virtual Boolean   HandleKeyPress(
    const EventRecord &inKeyEvent );
  virtual Boolean   ObeyCommand(
    CommandT inCommand, void *ioParam );
  virtual void    FindCommandStatus(
    CommandT inCommand,
    Boolean  &outEnabled,
    Boolean  &outUsesMark,
    Char16   &outMark,
    Str255   outName );
  virtual void    DrawSelf();

protected:
  char       mChar;
};
```

- Save your typing and close the window.

The `CSingleCharPane` class is derived from both `LPane` and `LCommander`. The class definition starts off by creating the enumeration constant class_ID which has a value of `'Cmdr'`, the same value you typed into the LPane's Class ID field in Constructor. Next comes all of the member function declarations and, finally, the declaration of the data member `mChar`.

Editing BeepCommander.cp

Your final bit of work is to add a few lines of code to `BeepCommander.cp`.

- Open the file `BeepCommander.cp`.
- Add these lines to `ObeyCommand()`, just after the call of `LWindow::CreateWindow()` and just before the call to `theWindow->Show()`:

```
CSingleCharPane *theCharPane =
  (CSingleCharPane *)theWindow->FindPaneByID( 2000 );
theWindow->SetLatentSub( theCharPane );
```

- Add this line to top of the file at the end of the `#include` list:

```
#include "CSingleCharPane.h"
```

- Go to the top of the file and change the const `window_Sample` to have a value of 1000, like this:

```
const ResIDT window_Sample   = 1000;  // EXAMPLE
```

- Finally, add this code to the constructor:

```
URegistrar::RegisterClass( CSingleCharPane::class_ID,
  CSingleCharPane::CreateSingleCharPaneStream );
```

Till Next Month

Well, that's about it for BeepCommander. Once all your code is in, run the darn thing. The window shown in Figure 1 will appear. Type some characters and watch the letters flash by. Notice that the **Special** menu is enabled only when you type the letter 'x'. Why is the entire menu disabled and not just the **Beep** item? This is a feature, not a bug. PowerPlant disables a menu title when all of its items are disabled.

Next month, we'll expand our horizons a bit more and explore yet another corner of PowerPlant. See you then...

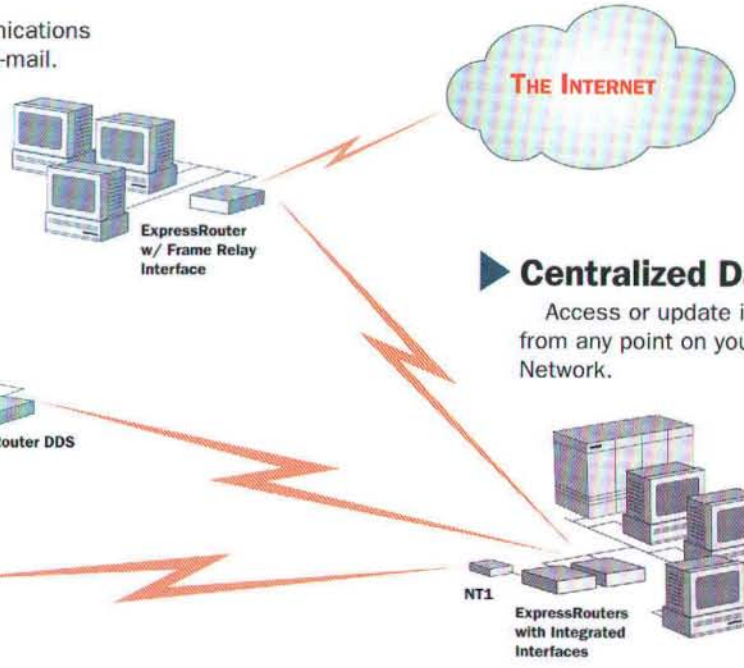
Fast remote internetworking in a *single* box.

Shared Access to the Internet

Your whole department or organization can connect to the Internet—full time or just when you need to.

E-mail

Streamline communications with enterprise-wide E-mail.



Telecommuting

High speed access to the entire office network.

Centralized Databases

Access or update information from any point on your Wide Area Network.

Videoconferencing

Close the gap between distant sites with LAN-to-LAN videoconferencing.

Fast File Transfer

Transfer files at speeds up to 1.544Mbps—10 MB per minute.

What makes the ExpressRouter the best choice for your Wide Area Network (WAN) needs?

- Built-in WAN interfaces for Frame Relay , ISDN, T1, and 56Kbps DDS.
- Expandable architecture that supports up to 3 high-speed WAN ports.
- Built-in support for AppleTalk, TCP/IP and Novell IPX LAN protocols.
- Interoperability via PPP.
- Support for SNMP and Telnet management standards.
- Powerful, easy-to-use auto-dialer software designed for the Macintosh.

Engage Communication, Inc.

9053 Soquel Drive • Aptos, CA • 95003

Tel: 408.688.1021 • Fax: 408.688.1421 • E-mail: sales@engage.com •WWW: www.engage.com

By Don Crabb

Putting the Moves on the Internet

Well, we've all wondered, of course. Wondered when Microsoft would get serious about the Internet. Serious about leading its development. Serious about co-opting it as another market in the Microsoft hegemony. Serious about making the Internet and the World Wide Web the The Next Big Thing.

Well, on December 7, 1995 (is it just me, or does that date send shivers up your spine, too?), Microsoft stopped our wondering and let us have it – right between the Web pages, if you will.

To wit, Microsoft has decided to stop fooling around with its Internet science project (Microsoft Network) and to start making "every effort" towards "dominating" the World Wide Web, according to Microsoft co-founder and chairman, William Gates III. As such, Microsoft plans to integrate Internet components that it is developing, as well as those it will license from others, along with communications links, into *every* Microsoft product, including Windows 95 and NT, Microsoft Office, Visual Basic, Access, and all of its development tools, including SQL Server and other big ticket items. I suppose it will even get into an upcoming release of Microsoft Bob!

The Microsoft Effect

Whether Microsoft can come in and repeat its 800-pound-gorilla act among Internet and Web vendors remains unclear, however, thanks, in no small part, to the current Web hegemony of NetScape Communications Corp. and the heavy Web bandwagon that Apple has successfully pulled down the road for most of 1995.

But just the mere mention of that 800-pounder stomping around the Web caused NetScape's stock to drop 18 percent in a single day of trading – down $28.75 to $132.50 on December 7. Another Microsoft Net competitor, Netcom On-Line, lost $8.75 to close at $54.75 that day. Ouch, that hurt!

Showing the ripples that the big hairy beast makes no matter where it treads these days, shares of Microsoft partner Spyglass also fell that same day, dropping $14.75 to $95.25 – despite the fact that Microsoft's new Internet presence will make it heavily dependent on technology that Spyglass is developing for its Internet Explorer browser.

Meanwhile Microsoft's own stock slid a piddling 12.5 cents to close at $90.50.

Besides partnering more heavily with Spyglass (watch your backs, guys), Microsoft has embraced Sun Microsystem's Java and JavaScript, a mere week after every other computer company on the planet already declared JavaScript the new lingua franca of the multimedia Web. Apparently, even Microsoft is mortal.

Although we can certainly expect them to figure some way to co-opt Java to their own market aspirations – no matter how "open" Sun insists it will remain (just ask CI Labs how much Microsoft has embraced OpenDoc if you doubt their comfort factor at dealing with standards they don't own and control) – with Java already being supported by all the other computer bigshots (including Apple), Java's probably too big for even

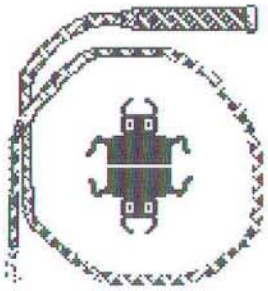
Don Crabb – Don is a contributing editor and columnist for *MacTech, MacWEEK, MacUSER, Mac/Chicago, Digital Chicago, MacToday, Win95User, ComputerUser, The Chicago Sun-Times Features Syndicate, The Springfield Union-News, PC Magazine,* and about a million other publications. Don welcomes comments at his Internet address: don_crabb@mactech.com. You can also check out his WWW Home page at http://www.cs.uchicago.edu/~decc/.

Gives you the Information to *Program your Best*!

The Debugger V2 & MacNosy

by Steve Jasik

Information



Control

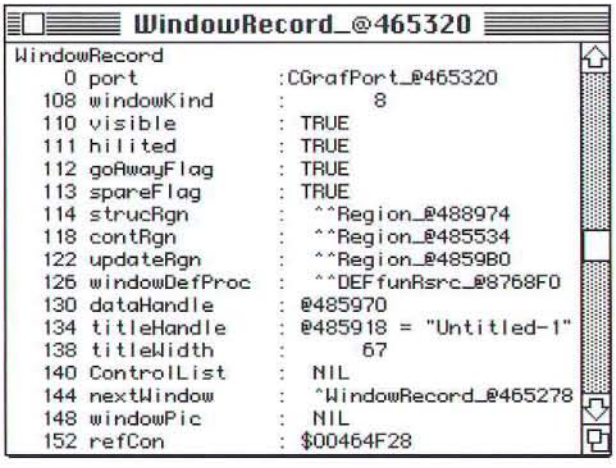
The Debugger is a low and high-level symbolic Debugger that runs in a full multi-window Macintosh environment. You can trace program execution, view the values of variables, etc. ***of both 68K and PowerPC programs***.

MacNosy is a global interactive disassembler that enables one to recover the source code of any Mac application, resource file or the ROM.

When you compare features of the different debuggers, note that *only one* has all the below features to help you get your job done, and *only one* has MacNosy to help you debug any program in a full system (6.0x or System 7.x) environment symbolically!

It is the *only* debugger to use the MMU to protect your CODE resources and the rest of the system from the program you are debugging. With MMU Protection you can find errors when they happen, not millions of instructions later! (Macintoshs with 68030 CPUs only).

The Debugger is the debugger of choice at: Adobe, Aldus, Claris, Electronic Arts, Kodak, Metrowerks, etc.



An example of a structured data display window

Its Features Include:

- **Symbolic Debugging** of *any* Macintosh program, ROM, or *code resource* (DRVRs, XCMDs, INITs, PDEFs, 4DEXs ..)
- **Source level debugging for Metrowerks & MPW** compiled programs (C++, C, Pascal, Fortran, ...), and an Incremental Build System with instant Link for superfast development.
- **Object Inspector for MacApp 3 programs**
- **Source level debugging of Think C™ projects**
- **Includes a program (CoverTest) to interactively do Code Coverage analysis for SQA testing, etc.**
- **Simultaneous Symbolic debugging of multiple "tasks"**
- **Fast Software Watchpoint** command to find clobbered variables
- **Sophisticated error check algorithms** such as Trap Discipline (Argument Checking), Handle Zapping, Heap Scramble and Heap Check to detect program errors before they become disasters
- **Structured display** of data (hypertext) with user definable structures while debugging
- **Conditional breakpoints** to help filter out redundant information
- **Continuous Animated Step Mode** to watch your program execute instruction by instruction
- **Detailed symbolic disassembly for both 680x0 and PowerPC** with symbol names, labels, cross ref maps, – make it possible to ferret out the secrets of the ROM, etc.
- **"Training Wheels"** for the PowerPC disassembler to help you learn the opcodes

The Debugger V2 & MacNosy: $350

Runs on all Macs. Call For Group prices or Updates. Visa/MC Accepted.

Available from: Jasik, APDA, Frameworks or ComputerWare (800-326-0092).

Jasik Designs • 343 Trenton Way, Menlo Park, California 94025 • **(415) 322-1386**

Internet: macnosy@jasik.com • Applelink: D1037

Uncle Bill to kill. No matter how much he'd like to, since it poses a big threat to the growth of Visual Basic as the choice for Web creation.

In the same set of announcements, Microsoft's Gates said that "the Internet is the primary driver of all new work we are doing throughout the product line." Gates also said, "We are hard-core about the Internet." As if any of us needed to be reminded. I can't imagine any Mac developer actually believing Microsoft would attempt anything less with the Net than it's done on the desktop – near-total hegemony.

So, there we have it. Microsoft is going to try to kill NetScape and as many other Net rivals as it can (IBM's Lotus Notes division, America Online, and Unix companies including Sun Microsystems – no matter what they say about Java). And those it cannot kill, it will try to absorb, like Spyglass.

What does this all mean for Mac developers?

The Mac and The Web and Mr. Bill

Well, make no mistake about it, Microsoft is psyched to chomp up the Net. According to Roger B. McNamee of venture capital firm Integral Capital Partners of Palo Alto, CA, "It was really fun to see Bill Gates so pumped up. He had the same kind of fire in his eyes that I suspect he had 20 years ago when he started the company.

"I imagine he has been wrestling with the approaching maturity of his core business, and wondering what to do with it," McNamee continued. "The Internet offers the right scale of business opportunity at just the right time."

But just because Bill and Microsoft are jazzed about devouring the Internet, does that mean they will pull it off? Not by a long shot. The differences between having a good plan and executing it flawlessly have slain bigger dragons than Microsoft. And with the Internet, where part of every successful software vendor's strategy to date has been free software, Microsoft has little practice.

Still, Microsoft has promised to give away its Internet Explorer and Web server software for most platforms – Unix, Windows 95, Windows NT, and even the Mac OS – in an effort to blow away NetScape and gain the lion's share of the market.

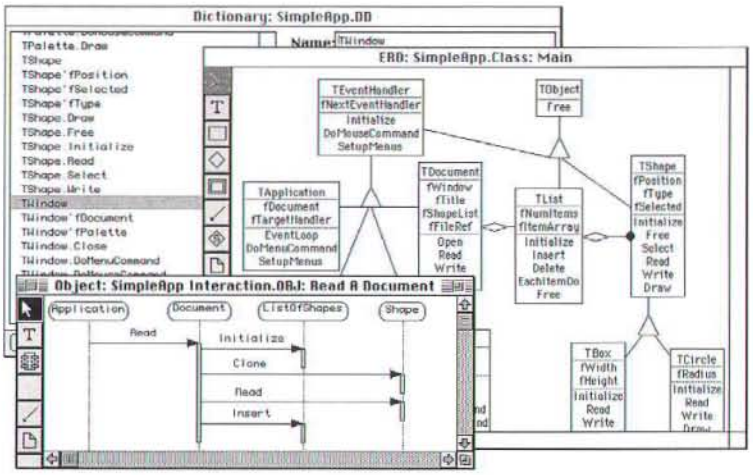
No matter how successful or unsuccessful Microsoft is at its Web hegemony strategy, Microsoft's move into the mainstream of the Net and the Web is good news for Mac developers. But only as long as Apple can continue to try to push the Mac OS into all the expanding corners of these markets (by getting us Cyberdog and OpenDoc, and by partnering with NetScape to provide us with a browser construction kit that can easily be incorporated into our own applications). If Apple can maintain its thrust, having Microsoft help out by pushing its own browser onto the Mac OS is a win for us.

The key here, though, is not what Microsoft does, but what Apple does (and what we do with it), as well as its major development partners. With Metrowerks cranking out a Codewarrior for JavaScript, we'll have one of the core tools we need to accelerate Web product development beyong the HTML and CGI state, but we need others that do not require C and C++ expertise.

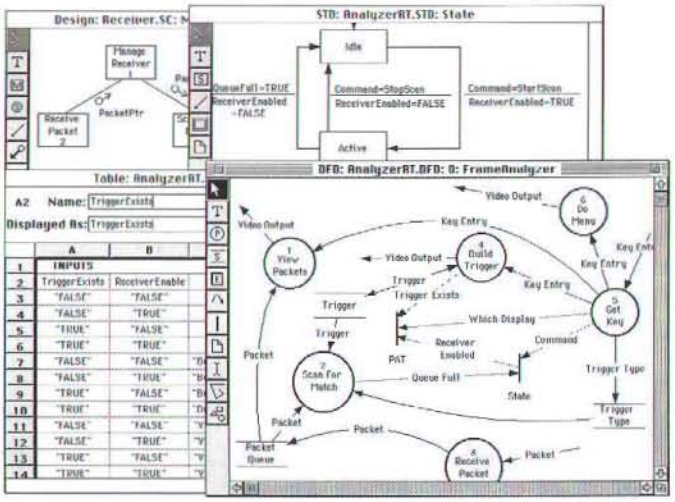
For Apple to help us carve out the Web high ground, we need tools that let AppleScripters (borrowing from Cyberdog in some ways?) build Web objects that work with the other tools they use each day, be they multimedia, or database, or spreadsheet, or decision support. In short, we have a real opportunity here, now that Microsoft has joined the Web party (and before they can choke it into proprietary conformity), to spawn a new generation of object tools for Mac users and managers that can be used without programming experience to build sophisticated query and information delivery systems on top of their Web servers, while also being able to build *ad hoc* looks into other Web sites. But they need the solid Mac tools to do this; HTML and CGI and even JavaScript are at once too limited, too complicated, and too tweaky.

Maybe Apple ought to reconsider Apple Dylan as the core for such a group of Web tools. Maybe SK8 could have a second coming as the engine behind such a tool package. Or maybe even a new version of HyperCard, strengthened with AppleScript, Cyberdog, and OpenDoc, and with an easy way to add Java applets, could fill the bill. I just hope that the bill gets a close examination while we still have this golden opportunity.

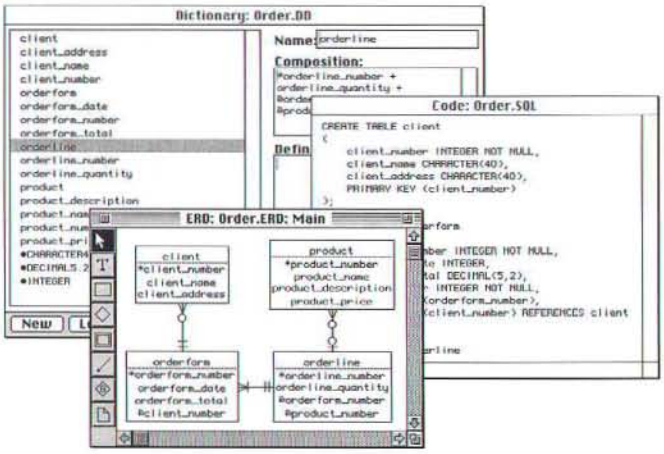
Draw and verify your design, generate code or reengineer existing code back to diagrams.



OOA/OOD includes OMT, Booch 94, Coad/Yourdon, Shlaer/Mellor...



Structured A&D using Yourdon/DeMarco, Gane/Sarson, Hatley/Pirbhai...



Data modeling and SQL generation for Information Engineering, Chen...

MacAnalyst
&
MacDesigner

Software Engineering

Structured Analysis & Design
Object-Oriented Analysis & Design
Real-Time & Multi-Task Design
Data & Screen Modeling
Integrated Code Editing & Browsing
Multi-User Dictionary & Requirements
Code to Design for C++, Pascal, Fortran...
Design to Code for C++, Pascal, Fortran...

New in Version 5.0

Jacobson, Fusion, Use Cases, Harel Statecharts, Interaction Diagrams and more

Products are available by single, site or educational license and supported with on-site training, update service, free newsletter and technical phone support. Products include sample documents, tutorials and online help.

Product Options

MacAnalyst	$ 995
MacAnalyst/Expert	$1595
MacDesigner	$ 995
MacDesigner/Expert	$1595
MacA&D	$2995
Translator	$ 495

System Requirements

Macintosh System 7 with 8 meg RAM or UNIX with Apple's MAE software. Winner of the 1994 CIO Magazine Readers' Choice Award, MacAnalyst and MacDesigner are used by thousands of developers worldwide for personal computer, mainframe and embedded software projects.

Call now for free technical brochures!

Excel Software

515-752-5359

P.O. Box 1414 • Marshalltown, IA 50158
casetools@aol.com • Fax: 515-752-2435

FREE INSTALLER!

To: Mac Developers & Product Managers

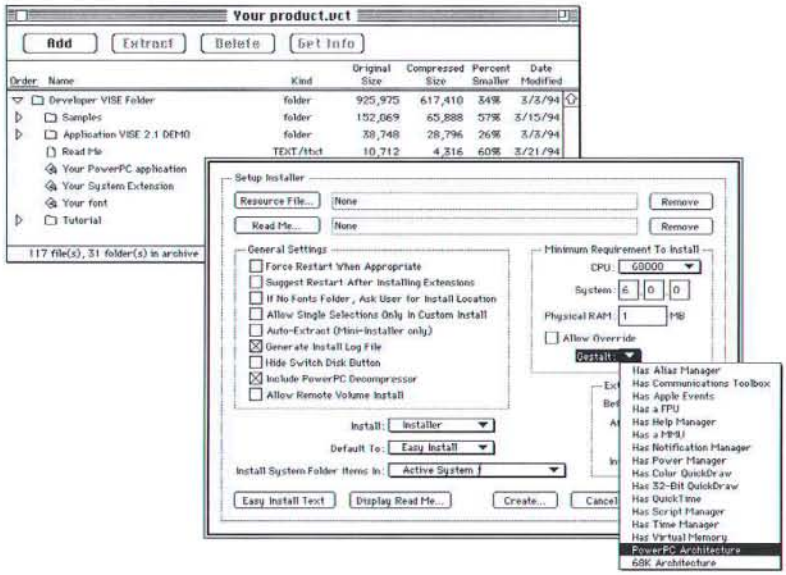
From: MindVision (Co-authors of Speed Doubler)

Subject: Free Copy of DEVELOPER VISE

Message: We've got a great new installer for you.
Here's your chance to try it for free.
No risk, no obligation, no hassle.
Call today, don't delay!

Full PowerPC Support

Integrated compression



No programming required

Fully graphical interface

www.mindvision.com

INTERNET: SALES@MINDVISION.COM • APPLELINK, AOL: MINDVISION • COMPUSERVE 70253,1437

Join the growing list of companies who use our VISE technology: Adobe, America Online, Apple, Claris, CompuServe, Connectix, Intuit, MacroMedia, Netscape, Symantec, WordPerfect, and hundreds others.

MindVision Software

MindVision Software 840 South 30th St., Suite C, Lincoln, Nebraska 68510
Voice: (402) 477-3269 Fax: (402) 477-1395 Internet: sales@mindvision.com
AppleLink, AOL: MindVision • Compuserve: 70253,1437

VISE COMPRESSION TECHNOLOGY

By Michael Hopkins, Symantec

SYMANTEC.™

This monthly column, written by Symantec's Technical Support Engineers, is intended to give our readers technical information on using Symantec products.

Q: I am compiling my project with the THINK Project Manager and I get an error message `"Error: Illegal Near Data"`. What can I do to correct this problem?

A: This typically occurs when you have **Far DATA** checked in the **Project Options** dialog box and not **Far CODE**. To turn on Far CODE, choose **Set Project Type** from the **Project** menu and check the **Far CODE** check box. For more information on how to use Far CODE and Far DATA, consult your User's Guide.

Q: I have written some simple code that has a structure declaration in a header file and then a global variable of that structure type in my main source file. When I compile the code, I get the message `"Error: Size of struct is not known"`. Furthermore, when I preprocess the file, I don't see my `struct` declaration being included. What is going on?

A: Let's take a look at an example:

```
/* main.c */
#include "timer.h"

struct clock local_clock;

main()
{
}
```

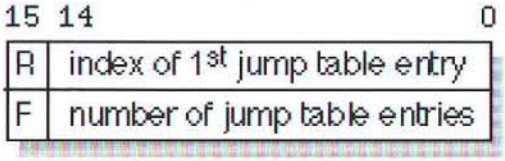
```
/* timer.h */
struct clock {
 long start_time;
 long end_time;
};
```

What is wrong with this picture? Well, there is actually nothing wrong with the structure. The problem is that `Timer.h` is the name of a system include file that is automatically included as part of the precompiled headers (in this case, `MacHeaders`). Therefore, the user header file is not being included because a system file of the same name is being used. To correct this problem, either rename the user header file or, for non-Macintosh applications, do not include `MacHeaders` in the prefix.

Q: I am writing a virus scanning program and I need to examine code resources of an application to verify that they are valid. What information does the Symantec Linker place in the first two bytes of the code resource?

A: For all `CODE` segments besides `CODE 0`, there is a code segment header. The THINK Linkers use the upper bit of this header to indicate a model Far CODE segment. The runtime loader resides in `CODE 1` of the application and is the first piece of code executed. The loader loads and initializes the `DATA` and `STRS`, installs hooks for `_LoadSeg`, `_UnloadSeg`, and `_ExitToShell` traps, and calls the main program.

If the code is using a far model, the `_LoadSeg` and `_UnloadSeg` bottlenecks completely replace the standard segment loader. The standard 4-byte `CODE` segment header is interpreted differently to accommodate the larger jump table, so it is incompatible with the ROM segment loader. The header has the following format:



The *R* bit indicates that the segment has relocations which must be applied at runtime. These are stored in a `CREL` resource with the same resource ID as the `CODE` segment. The *F* bit is used to distinguish a far header from the standard header.

Be aware that this format is different from the header that MPW and Merowerks use as well as the CFM-68K header format.

Q: I am trying to use ODBC (Open DataBase Collaboration) with the THINK Project manager and I am getting a number of link errors. What library files to I need to add to use ODBC?

A: To use ODBC, you will need to include:

- `MacTraps` and `MacTraps2`
- `LibraryManagerClient.o`
- `alloc_private.c`

You will also need to write two additional functions (source is in THINK Reference under Embedded DefProc):

```
Boolean TrapAvailable( short theTrap );
void FlushCache( void );
```

If you are compiling one of the demos, you will need to write a `stricmp`:

```
pascal short _stricmp( const char *s, const char *s2)
{
    return (short)strcmp(s, s2);
}
```

Q: I am making a 68K version of my PPC application and I'm getting a link error with `numtostring()` which is in `TextUtils.h`. On the PPC, this routine is in `InterfaceLib.xcoff`. Where is it on the 68K mac?

A: It is important to realize that the lower-case version of `numtostring()` is different than `NumToString()` which is in `MacTraps`. `NumToString()` returns a pascal-style `Str255`, and `numtostring()` returns a c-style string. To use the lower-case version with the THINK project manager, you will need to include `Apple C Glue` which is in `Macintosh Libraries:68K Libraries` from the *Symantec C++ v8 Release 4* CD.

Q: Is there any way that I to make the Finder run Power Macintosh DebugServices when I launch the SPM?

You Can't Wait Forever.

Let's Face It.

As a professional developer or webmaster, you have to think of yourself first. You have deadlines to meet, products to deliver, and people depending on you.

A Tool in Hand...

Your tools are your livelihood. You need up-to-date technologies and timely support. You can't afford to wait for vaporware to materialize or for your tools vendor to leisurely add features that you need today.

When Hell Freezes Over?

When was your text editor last updated? Six months? A year? Five years? And when you contact the vendor that produced it, do they tell you how they're working on the next great version, how wonderful it's going to be, and how it will ship "Real Soon Now"? Or worse yet, do you get a disconnected phone number?

That's where we come in.

At Bare Bones Software, we deliver the best tools here and now. We have a distinguished tradition of continually enhancing the functionality, performance, and design of our products to meet our customers' needs. We don't allow our products to languish until we feel like doing something about it. We won't hype vaporware as the solution to today's problems. And we haven't drifted in and out business on a whim. **We're here to support you for the long haul.**

What About the Software?

You don't have to wait for the software, either. We're constantly refining our techniques and technology to bring you the fastest and most efficient tools. As the result of our efforts, BBEdit has repeatedly raised the standard for Macintosh text editing. (In fact, we've done so many cool things, several of our competitors have tried to copy us. Imitation is the sincerest form of flattery, after all. We're *very* flattered.)

Why Believe Us?

Since its commercial debut in 1993, BBEdit has set the pace for capability, user-interface, functionality, and customer support that others have yet to match. BBEdit was the first text editor to support ToolServer, THINK Reference, Toolbox Assistant, THINK C/Symantec C++, Internet Config, Quickdraw GX printing, PowerTalk, PowerPC acceleration, AppleScript, Apple Guide, and Macintosh Drag and Drop. We're still setting the pace for others to follow, as the only Macintosh text editor to integrate directly with Metrowerks CodeWarrior, through the use of the ClickWarrior extension.

See for Yourself.

We've painted a great picture, but don't take our word for it. Visit our Web site and see for yourself. You'll find lots of things to like.

BBEdit. Because you can't wait forever.

Bare Bones Software

Bare Bones Software, Inc.
P.O. Box 1048
Bedford, MA 01730-1048 USA

Voice: (508) 651-3561
Fax: (508) 651-7584
E-mail: bbsw@netcom.com
Web: http://www.tiac.net/biz/bbsw/

BMUG Choice Product FALL 1995

Accelerated for Power Macintosh

Mac OS

A: No, but you could tell SPM to launch DebugServices on startup or shutdown. To run a script automatically when the Project Manager opens, record or write a script using the Script Editor. Save the script in the (Scripts) folder and name it "Startup". Alternatively, if you want the script to run on exit from the Project manager, name it "Shutdown".

Q: I've noticed that all native applications have a note in the "Get Info" window that says: "Memory requirements will decrease by xxxxK if virtual memory is turned on in the Memory control panel". Won't using virtual memory decrease my application performance?

A: No, not necessarily. In some cases the use of virtual memory on Power Macintoshes can actually increase the runtime performance of your application. When VM is not enabled on a Power PC, the application's stack, heap, and all of its code fragments have to be loaded into the application's partition. With VM on, only the stack and heap are loaded into the application's partition. This reduces launch time and requirements for native programs. The Virtual Memory Manager will track your application's code fragments and load them into the application's partition only when they are needed. If done correctly, this won't result in a noticeable performance penalty and the application will launch much more quickly. For more information, refer to *Inside Macintosh: PowerPC System Software.*

Q: How do I create a `CustomTEHook` on a PowerPC?

A: It is actually fairly straight-forward thanks to Apple's Universal Headers. Create your hook procedure with the following signature:

```
pascal unsigned short myTextWidthHookProc(
    unsigned short  textLen,
    unsigned short  textOffset,
    void *          textBufferPtr,
    TEPtr           pTE,
    TEHandle        hTE
);
```

Then create a routine descriptor and call `TECustomHook` like this:

```
TextWidthHookUPP myUPP =
    NewTextWidthHookProc( myTextWidthHookProc );
TECustomHook( intTextWidthHook, &myUPP, myTE );
```

Q: I have noticed that there is a problem in `CDialogText` where cuts, copies and pastes are not reported to the supervisor of the `CDialogText`. Is there an easy way to fix this?

A: Yes, there is. Change the code for `CDialogText::DoCommand` to:

```
void CDialogText::DoCommand( long theCommand )
{
    inherited::DoCommand( theCommand );

    switch( theCommand )
    {
        case cmdCut:
        case cmdPaste:
        case cmdClear:
          if (editable)
            BroadcastChange( dialogTextChanged, &ID );
          break;
        default:
          break;
    }
}
```

Q: How do I force Visual Architect to re-generate all sources including the files that should only be generated once such as `CMain` and `CApp`?

A: Take the `Source` directory in your project folder and either rename it or move it to a different location. Remove all of your existing generated VA files from your project. When you choose **Generate All** from the Visual Architect, it will create a new source folder and generate all of the files and then add them to your project.

Special Thanks To

Levi Brown, Craig Conner, Rick Hartmann, Noah Lieberman, Andy McFarland, Scott Morrison, Phil Shapiro, Jeff Weeks, Kevin Quah

By Brian Arnold

MacApp 2 for PowerPC in Object Pascal

"Object Pascal is not bad, it just smells funny." — apologies to F.Z.

Whither Native?

This is an article for developers who have Object Pascal MacApp 2 code and want to make that code native for PowerPC *[or for anyone whose pulse was set racing by Brian's and Guy MacCarthy's preliminary report in the November 1995 issue – man].* Along the way, it will describe the MacApp2PPC developer's cooperative as an example of how developers can help themselves when they need it the most. Although the MacApp framework itself has long since made the switch to the PowerPC, it has done so at the expense of Pascal Object support. The latest version of MacApp, 3.3, is written in C++, and has features to make you salivate – drag and drop, AppleScript, PowerTalk mailers, performance optimizations – and version 3.5 will have OpenDoc container support. You could convert your source code and upgrade today; many developers have done this. There are a number of excellent Object Pascal to C++ translators available on the market to help you in this conversion. But if you're resource constrained, or C++ syntax causes a gag reflex, or you have some other Very Important Reason to stay with your Object Pascal MacApp 2 code, MacApp2PPC provides another alternative.

MacApp 2 for PowerPC

The Pascal version of the MacApp framework is hardly five years old now, and it's already doomed to obsolescence. Or is it? Maybe once OpenDoc and SOM arrive in full force, we can seal the coffin on this version of the framework and look elsewhere for Object Pascal framework solutions. But until then, the lean and mean MacApp 2 still has some mileage left in the millions of lines of excellent Macintosh code written using it.

So what's so difficult about porting MacApp 2 to PowerPC? To find out, I announced a developer's cooperative at the Apple May 1994 WWDC. Immediately, twenty developers joined. We encountered several difficult obstacles, and by distributing the effort among the developers, we were able to make the necessary changes to MacApp 2 in just under 12 months.

Of course this would all be academic if there hadn't been support from Apple and the compiler vendors. Language Systems (LS) licensed MPW Pascal from Apple and further developed a PowerPC back-end; LS is now developing it as a Symantec IDE plug-in compiler. They were instrumental in providing us with support when we were just getting started. Apple aided in the development of the project, particularly at the hack session that helped finalize the port. Metrowerks developed a compatible PowerPC Pascal compiler, and later added Pascal Object support and made it a plug-in compiler for the CodeWarrior (CW) IDE.

MacApp 2 for PowerPC has been designed to work with both the LS Pascal and CW Pascal compilers, and is provided by both Language Systems and Metrowerks for use with their native Object Pascal compilers.

Brian Arnold – Brian Arnold is the director of software development at *Lumina Decision Systems, Inc.* He's the chief architect of Analytica®, a visual decision analysis modeling tool, and he spends his spare time fiddling with OpenDoc. You can visit his web site at http://www.lumina.com/ or you can reach him at arnold@lumina.com.

MacApp 2 Conversion

Here's the short list of elements in MacApp 2 that had to be changed for PowerPC.

- MacApp 2 makes extensive use of 68K assembly and in-line code that can't run (easily) on PowerPC. This code had to be rewritten in Pascal, C, or PowerPC assembly.
- MacApp 2 relies on AppFile Toolbox calls and has no support for required Apple events. The PowerPC runtime architecture does not support AppFile calls for opening documents at application startup, and expects you to support the required Apple events (open application, open document, print document, quit application).
- MacApp 2 uses 68K-specific methods for trap patching. On PowerPC, patching a Macintosh Toolbox trap is a lot less trivial and more costly, so it needs to be done differently and less often.
- MacApp 2 makes extensive use of ProcPtrs for Mac Toolbox callbacks. Since the PowerPC Toolbox is designed to operate with both PowerPC and 68K applications, all callback ProcPtrs need to identify whether they are compiled for 68K or PowerPC. A toolbox glue routine for each ProcPtr fills in this information, plus some other housework, and returns a UniversalProcPtr that you use instead of the ProcPtr.
- MacApp 2 uses old and obsolete Toolbox names extensively. The PowerPC runtime architecture uses shared libraries (code fragments) that implement newer, more consistent names for Toolbox calls. Because Toolbox calls are resolved by name, the new Toolbox names must be used. In addition, the Pascal QuickDraw globals `thePort`, `gray`, etc., now reference the `qd` global (e.g., `qd.thePort`), as it is in C.
- Developers have relied on unsupported extensions that also needed to be ported. These extensions include Model Far, floating windows, tear-off menus, and synchronized scrolling.
- Finally, adding support for PowerPC Object Pascal compilers and IDEs was a prerequisite. The build process needed to be revamped to support the latest development environments and tools, and the documentation had to be revised.

This sounds a bit daunting. Frankly, when I announced the MacApp2PPC developer's cooperative, I didn't have the foggiest idea whether or how we were going to succeed. However, I did know a little bit about what needed to get done and I called upon members of the MacApp 2 developer community for their interest in helping themselves. I did not expect this inquiry to generate a lot of interest, but we developed into a 100-member Internet mailing list. I also did not expect to witness much enthusiasm, but it was there in spades.

Essential Ingredients for Making It Happen

We were so successful with the MacApp 2 PowerPC port that I am compelled to muse on why this was so. It isn't so hard to identify the elements, because we promoted them from the start.

- Nobody else is going to do this for you, so you had better do it yourself.
- Only do what is necessary, and do no more.
- Don't change anything that doesn't need to be changed.
- Trust your instincts.

When you take personal responsibility for managing a task, the details become much clearer to you, and you are more productive. Secondly, when you focus on only what is necessary, the right thing to do comes out of the woodwork and other gunk tends to stay out of the way. Finally, when you don't change things that would seriously affect the user of the framework, something mysterious and wonderful happens: their code gets ported faster. The extra ingredient, "trust your instincts," is the reason why I persisted with MacApp2PPC for the past year and a half. It didn't hurt that the Internet (and AppleLink) helped people across continents share ideas and code to make this happen.

For the first eight months, Masahiro Abe, Per Bergland, and Dave Johnston laid the foundation for the port by switching MacApp to the new Toolbox names, rewriting assembly code in C, Pascal and PowerPC assembly, and developing conversion scripts for our own code, while the remainder of the cooperative provided feedback.

In March 1995, when it looked like we had completed all the difficult porting tasks, we held a three-day hack session at Lumina, where a dozen developers and Language Systems and Apple engineers converged to identify the remaining porting issues. Lumina's Brendan Del Favero set up the ethernet, and John Corbett made the logo. LS's Steven Hopkins and Steve Lavagnino were in the compiler hot seat. Apple's MPW entomologist Greg Branche helped out with MPW-related issues. Dave Johnston practically lived inside the PowerPC debugger all weekend. Cheryl Lins and I went wild on adding floating windows support to MacApp 2. Larry Tesler, Apple's chief scientist and the one to blame for MacApp 1.0, stayed up late hacking code with Colleen Barton, and contributed a change to his USynchScroller unit. Andrew Peterson and Larry Hamel came to port their MacApp 1.0 application. Eric Jackson, Mr. MacWireFrame, didn't use MacApp 2, but provided a lot of feedback on PPC Pascal code generation. Masahiro Abe, Per Bergland, David Shillitto, and Ingo Ciechowski provided remote input and feedback. As you would expect for such an event, Prograph guru and MacApp afficionado Kurt Schmucker visited and handed out C++ barf bags. I highly recommend throwing a hack session; nothing helps better to identify the critical tasks while having a lot of fun.

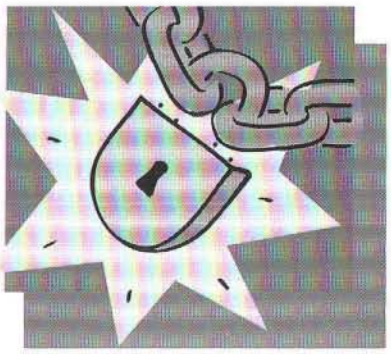
A few weeks after the session, Metrowerks added their support with the help of CW Pascal architect Marcel Achim and developers Roger Brown, Peter Baum, Donald McCormick, Farhad Anklesaria, and Scott Vorthmann, all of whom contributed one change or another. By the May 1995 Apple WWDC in San Jose, CA, we were able to claim that "Nothing

dtF The Relational Database System

dtF v1.6 including numerous SQL enhancements and ODBC driver available!

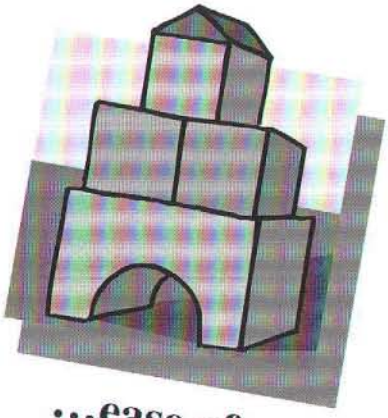
...performance

When it comes to performance **dtF** is in a class all its own. **dtF** utilizes a proprietary query optimization and caching scheme to obtain unparalleled performance. If you are in need of a quantum leap in performance, use the **dtF** native PowerPC stand-alone or client/server version.



...data security

Full transaction control and automatic level-3 error recovery guarantee maximum data protection even after sudden system crashes. **dtF** databases are compressed and encrypted to protect against all unauthorized access, even disk editors. Implement sophisticated table and operation level security with **dtF**'s password features.



...ease of use

With **dtF**'s unique direct integration technology, the **dtF** database system is fully contained in your application, enabling you to build double-clickable database applications. Never worry about missing or conflicting INITs or drivers. **dtF**'s native API is compact, easy-to-use and integrates seamlessly with all major development environments on the Macintosh. **dtF**'s high performance SQL, integrated data dictionary, security features, automatic index selection, query optimization and error recovery allow you to concentrate on creating great database applications instead of messing around with the internals of the database system.

Stand-alone applications built with dtF are royalty-free!

dtF is available for Macintosh System 7.x (68K and native PowerPC). **dtF** supports MPW C/C++, Symantec C/C++, Metrowerks CodeWarrior (all compilers 68K and native PPC). **dtF** is fully OpenDoc™ compatible. Separate versions for use with HyperCard 2.x, SuperCard 2.x, Smalltalk-Agents and Pictorius Peregrine are provided. AppleScript interface via DataScript™ for **dtF** from General Knowledge. **dtF** supports cross-platform development on Windows 3.11, Windows 95 and OS/2.

dtF Americas, Inc.
19672 Stevens Creek Blvd.,
Suite 128
Cupertino, CA 95014
USA

Phone: (800) DTF-1790
Fax: (510) 828-8755
AppleLink: DTF.AMERICA
Internet: dtF.america@applelink.apple.com

dtF The Relational Database System

works!", "Failure fails!", and "Windows float!" In case you're not familiar with MacApp, Nothing is the name of the 4-line "Hello World" example application, UFailure is the name of the seminal framework failure handling mechanism, and windows never used to float in MacApp 2 (that part we borrowed from MacApp 3). These were our milestones. Most of us were amazed that we succeeded in one short year.

Porting Your Source Code

Along the way to developing MacApp2PPC, we tried to come up with tools and hints for making the leap to PowerPC with your own code. The result is a set of MPW scripts that you can execute to do most of the conversion work for you. However, because MacApp2PPC was constructed to minimize the damage to your own code, you may be surprised at how little time it would take you to port even without the MPW scripts.

Running the scripts is easy, provided you can already tolerate MPW.

1. Install MacApp2PPC and a minimal MPW according to the instructions provided on your compiler vendor's CD. It is best to use a fresh MPW setup, not one that you're already using.
2. Start up MPW. Confirm that MacApp2PPC installed correctly (its menu is added to the menu bar).
3. Set the current directory to one containing your Pascal source code, and select **Convert Directory** from the MacApp2PPC menu. Several dictionary files are run against your code using the MPW Canon tool. This performs the majority of the Toolbox renaming changes.
4. You can run a script called CreateUPPChecker to find out where all of your ProcPtrs live, because you will have to change them to create UniversalProcPtrs. Because this script takes a long time to run, it's best to start it at night and return to it the next day (or the next, depending on how large your application is).

By this point, most developers cut to the chase, try to build, and wade through the remaining compiler and linker errors. We'll cover a few of them here.

Changing those Pesky ProcPtrs

One common hang-up in the migration to PowerPC is the need to replace ProcPtrs with UniversalProcPtrs for Toolbox callbacks. We'll take a moment to describe how to change a ProcPtr through an example in MacApp 2 itself.

The process goes like this: you have identified a ProcPtr that must be changed. You need to create a UniversalProcPtr for that procedure pointer, and use that UPP in its place. The UPP is a record that includes your procedure pointer, information about whether it's compiled for 68K, and other housework. If you use that UPP repeatedly or in a modeless manner (for example, a TextEdit `clickLoop` callback), you might hang on to it; otherwise, you will usually dispose of the storage used by the UPP right after making your Toolbox call.

For example, in MacApp 2 for PowerPC, when Finder printing, we use the Apple event Toolbox call `AEInteractWithUser` to make sure we get the user's attention before putting up the print dialog box. This call has a callback to our own code, so that we can idle while the Apple event manager is off waving its hand in front of the user. Here is the interface to the idle function.

```
FUNCTION IdleFunction (
    VAR event:      EventRecord;
    VAR sleepTime:  Longint;
    VAR mouseRgn:   RgnHandle): BOOLEAN;
```

Before porting to PowerPC, the address of this function would be passed directly to the Toolbox:

```
FailOSErr(AEInteractWithUser(
    kAEDefaultTimeout, NIL, @IdleFunction))
```

For PowerPC, to create the UPP automatically, we go searching for the Toolbox file that contains the definition for the relevant function, in the hope of finding a definition for its UPP and an automatic conversion function. In this example, the Toolbox file is `AppleEvents.p`; lo and behold, there is an `AEIdleUPP` type and `NewAEIdleProc` conversion function provided for us. Because we no longer need the UPP when we are through, we will dispose of it when finished. So, in order to call `AEInteractWithUser`, we add a local variable and change the `AEInteractWithUser` code.

```
VAR
  idleUPP: AEIdleUPP;

  idleUPP := NewAEIdleProc(@idleFunction);
  FailNIL(idleUPP);
  FailOSErr(AEInteractWithUser(kAEDefaultTimeout, NIL, idleUPP));
  idleUPP := DisposeIfRoutineDescriptor(idleUPP);
```

When building for 68K, this changed code still works, that's why it's called Universal. In fact on 68K it does the same thing as before: `NewAEIdleProc` simply returns the `@`, and `DisposeIfRoutineDescriptor` does nothing. We are doing this extra work for PowerPC builds.

Working with fp and fenv

Another area of difficulty in porting your own code to PowerPC is likely to be the use of SANE or the 68881 (math coprocessor), if you do a lot of floating point math in your application. The good news is that the MPW porting scripts provided with MacApp2PPC make a lot of the changes for you to use the PowerPC MathLib through `fp.p` and `fenv.p`. Still, there are a few gaps.

For example, the `Num2Str` and `Str2Num` functions aren't provided by MathLib. This bothered key developers porting MacApp2PPC, so these are now provided in `UMacAppUtilities`. If you already include `UMacAppUtilities` or `UMacApp`, you can continue using `Num2Str` and `Str2Num` as is. Note that the string parameter was changed from `DecStr` to `Str255`.

MacApp 2 API Changes

Aside from new interfaces and methods, the only major API change is in `TApplication:GetActiveWindow`, which now requires a Boolean parameter similar to MacApp 3. `kFloatersOK` and `kFloatersNotOK` are defined for you. If you use this function, the returned window can be a floating window or a regular window. When in doubt about how you need it to behave, use `kFloatersNotOK`.

Other important API changes include accepting UniversalProcPtrs instead of regular ProcPtrs (consult the previous description of converting to UPPs for help in changing your ProcPtrs).

Finally, UPatch's mechanism for patching traps has changed to conform to the MacApp 3.1 mechanism, which is friendlier for PowerPC trap patching. If you use UPatch directly, you will have to change the way you patch traps. Consult UPatch and MacApp 3.1 documentation for details.

What's More

We wouldn't be able to call ourselves good hackers if there wasn't a surprise or two in the converted MacApp 2. One surprise is integrated support for floating windows. There is an unsupported trap-patched floating windows unit used by a lot of MacApp 2 developers, but nobody wanted to wade through the gunk in porting that code, which is buggy anyway. So, Cheryl Lins and I looked at MacApp 3.1's support for floating windows, and we back-ported those changes into MacApp 2. It was easier than it sounds. The result is cleaner, integrated floating window support in the framework, and no trap patching. Finally, code for Balloon Help was mixed in with the code that supports Apple events (which Anil Bajaj originally wrote as a patch to MacApp 2 using Keith Rollin's excellent PatchMaker).

Apple events in MacApp 2

Several API functions are added for Apple event support. Default support for the required Apple events `'oapp'`, `'odoc'`, `'pdoc'`, and `'quit'` are implemented in `TApplication`.

You can support Apple events very easily with MacApp2PPC. You first create a new `'aedt'` resource in your `<app>.r` file specifying the event class, event message, and a `CmdNumber` (for an example look at the `'aedt'` resource in the `MacApp.r` file). The next and only other thing you have to do is override the `TEvtHandler` method `DoAECommand` and handle the appropriate `CmdNumber` (just like you would in `DoMenuCommand`, etc.).

DoAECommand interface

```
{ If the handler can perform the apple event command it }
{ does so by either performing the command directly or }
{ by posting a TCommand. NOTE: Both UNIV Ptr's are }
{ actually a Ptr to an Apple event record }

FUNCTION TEvtHandler.DoAECommand (
        aCmdNumber:CmdNumber;
        message:    UNIV Ptr;
        reply:      UNIV Ptr): TCommand;
```

In addition, there are two utility routines (in `UMacAppUtilities`):

- `MissedRequiredAEParameters` checks the passed-in Apple event for any required parameters that we may have missed (called if handling Apple events, which should not have any parameter like the open application and quit events.)
- `ProcessAEDocList` extracts the list of documents from the Apple event's direct parameter and converts each document's `FSSpec` record to an `AppFile` record; then calls the passed-in routine (used to handle the open document and print document events).

Note: Your `'size'` resource must have its `isHighLevelEvents` flag set in order for your application to receive Apple events. All of the MacApp2PPC example applications have this set already.

Floating windows

Floating window support was integrated into `TApplication` and `TWindow`. If you wish to add floating windows to your application, all you need to do is call `InitUFloatWindow` to ensure that the `TFloatWindow` subclass isn't dead-stripped.

To make a window floating, make your window resource class name a `TFloatWindow` instead of `TWindow`, and add the following to your `.r` file:

```
#include "FloatWindow.r"
```

This adds the System 7.5 floating window `WDEF` to your application, so that you have floating windows on older Macintosh systems. If you've been reading this article and you're skeptical of the value of object-oriented frameworks, notice the number of lines of source code the developer must write to use floating windows (answer: one line, and that's just to trick the linker!).

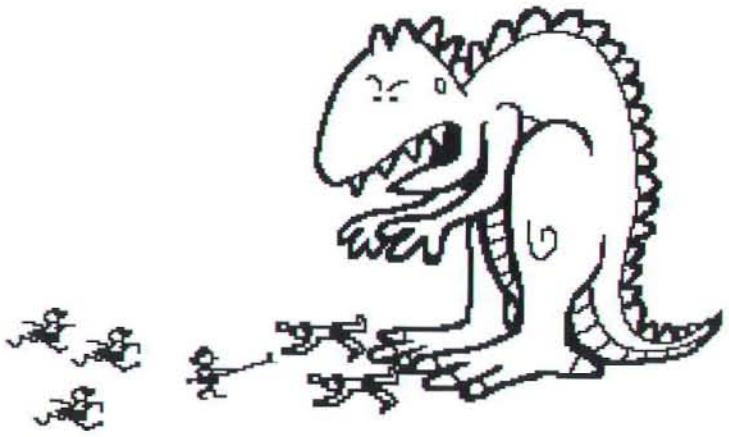
New Building Blocks

Actually, none of these building blocks are new, but they've moved from the MacApp examples into the main part of the MacApp library. If you are already using them, you will be mildly relieved to know they have been ported to PowerPC. You can remove them from your application source code, as well as from your MAMake file (if you're using MPW).

These building blocks include:

UBetterFeedbackCmd	VBL-synchronized feedback
UFloatWindow	Floating window support
UGrabberTracker	That MacPaint hand-grabber thingy
UMenu	TView descendants that can appear in menus
USynchScroller	Scrolling multiple views simultaneously
UTearOffMenu	What it says
UVUAssist	Virtual User support (OK, I lied, this is 68K only)

Are your projects eating you alive?



Are your projects eating you alive? When time is of the essence the quality of the people and their work means everything...

Swift performs above the standard level. We understand that experience, timing and costs can mean the difference for a successful project. When you need a person or a team to make things happen, call us.

We have provided solutions from the firmware level up. We work with the latest technologies such as Oracle Power Objects, OpenDoc/ODF, & Java to bring the latest options to solving your problems. Our technical services include Project Management, R & D, Training and Technical Support.

Swift can make the difference.
To see how, call (408) 338-1464.

S W I F T
CONSULTING
610 West Dr.
Boulder Creek, CA 95006

Compiler-Specific Details

It's worth mentioning one compiler-specific detail for each Object Pascal compiler.

Language Systems Pascal 1.0 works under MPW; that means it uses the MABuild tool to build MacApp applications. MABuild has been modified to accept additional options, including `-[No]PPC` to indicate 68K vs. PowerPC code generation. That's all there is to it! When a future version is provided as a plug-in for the Symantec IDE, the build process will use the features in the IDE more directly.

Metrowerks CodeWarrior Pascal works within the CodeWarrior IDE so the IDE is used for building applications, but MacApp 2 still requires compiling resources under MPW. Although there is a Rez plug-in for the CW IDE, MacApp 2 still relies on a custom MPW tool called PostRez. To make life easier for the CW Pascal user, the revised MABuild tool also accepts a `-[No]OnlyResources` option, to specify that you wish to compile (and PostRez) only resources for your application. This step is still annoying enough that Per Bergland and Roger Brown are seeking ways to remove or simplify the PostRez step, by developing a plug-in tool. Their dedication captures the spirit of what is MacApp2PPC.

And Finally

Well over a dozen applications have been ported to PowerPC using MacApp2PPC, and many more are currently being ported. Meanwhile, the world of software frameworks is constantly shifting. Component technology is moving rapidly to your neighborhood. MacApp2PPC is currently serving the needs of Macintosh Object Pascal MacApp 2 developers wishing to have native applications on PowerPC. Where it goes depends on the developers who use and nurture this version of the framework. There is no commitment except the one each developer gives to it. Some developers are sufficiently motivated to continue evolving MacApp 2 in manners consistent with the drive towards MacApp 3.3 as its flagship, and in moving developers toward OpenDoc as the future.

Developers in the cooperative are considering adding drag and drop support as an initial step toward OpenDoc container support, removing the working directories dependency, and removing the PostRez step mentioned earlier. If you want to make a difference and help out in making these changes, you can join the "MacApp2PPC-List" Internet mailing list by sending e-mail to macjordomo@afar.med.cornell.edu with "subscribe MacApp2PPC-List Your_Name" in the body of the message; send e-mail to arnold@lumina.com if you need help. Macjordomo (http://leuca.med.cornell.edu/Macjordomo) is a listserver that Michele Fuortes ported to PowerPC using MacApp2PPC in about 5 evenings. You are also highly encouraged to subscribe to the comp.lang.pascal.mac newsgroup on the Internet to stay in tune with the latest goings-on in Pascal for the Macintosh.

Also, check out the cooperative's Internet world wide web pages at: http://www.lumina.com/arnold/MacApp2PPC.html

You'll find some interesting things for MacApp 2 at this site, including the latest version of MacApp 2 for PowerPC, plus WASTE text engine support, QuickDraw GX printing support, and more.

What this cooperative brings Object Pascal MacApp 2 developers in the future is yet unseen. Stay tuned.

If you have a CD-ROM drive, then *you've gotta have* every article published in the first 10 years of MacTech Magazine!!

Available Now!!

... now in THINK Reference format!

MacTech CD-ROM, Volumes 1-10:

Over 1230 Articles.
All 115 Issues, including all of 1994.
All the Source Code.
THINK Reference 2.0.
Working Applications.
Full Documentation.
Demos for Developers.
And More!!

"When I designed THINK Reference, I envisioned endless databases at my fingertips. MacTech has doubled the information that is just a mouse click away."
– Darrell LeBlanc, Formerly of Symantec, Author, THINK Reference 2.0

"The CD strikes me as an impressive and very useful resource. The only disadvantage I've found is that each answer the databases yield exposes me to so many more issues, that I find myself exploring the articles for the sheer wonder of it, and thus putting off the real coding I should be doing. :-)"
– Nicholas De Mello MacTech CD Beta Tester

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

Voice: 805/494-9797 • Fax: 805/494-9798
Internet: custservice@xplain.com
eWorld: MT.CustSvc • AppleLink: MT.CUSTSVC
CompuServe: 71333,1063
America Online & GEnie: MACTECHMAG

✓ **50 MB of MacTech Magazine articles, Volumes 1-10**
Every article, 1230+ of them, from all 115 issues of MacTech Magazine printed from 1984 through 1994. Articles ranging from Assembly to BASIC, C to Pascal, Forth to FORTRAN and more. *And the articles are in THINK Reference!*

✓ **Hyperlinks to *Inside Macintosh* Databases of THINK Reference**
The articles have hyperlinks to relevant portions of the *Inside Macintosh* databases of THINK Reference. For example, if you are looking for information on Aliases, look at the MacTech articles on Aliases and use the hyperlinks to jump to the *Inside Macintosh* entry for the Alias Manager. And now, with the articles in THINK Reference, you can do free-text searches 8-10 times faster than you could previously with On Location™ 2.0.

✓ **100+ MB of MacTech Magazine source code examples, samples, and utilities.**
These are the files that go with the magazine – the code that the articles are talking about. Use them in your own applications, with no royalties!

✓ **A fully-capable version of THINK Reference**
Including *Inside Macintosh* Volumes 1-6 (7 MB).

✓ **Sprocket – MacTech's Tiny Framework.**
Build your simple application using a lean, mean application framework. Experiment with new code without having to build a whole application around it!

✓ **80 MB of FrameWorks, MacApp®, MADA and SFA articles, files and source code.**
The most complete set of FrameWorks archives known.

✓ **Apple APIs, Utilities, and SDKs**
Including: Universal Header Files, Discipline, Macintosh Drag and Drop SDK, MacsBug, Telephone Manager SDK, Thread Manager SDK, TrueEdit 1.8 and and more.

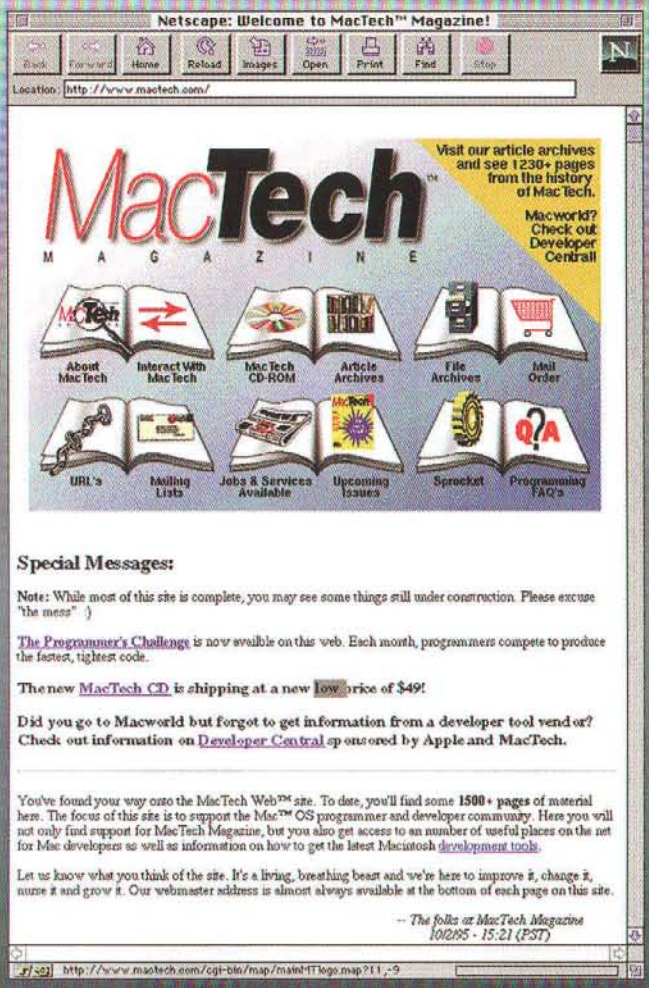
✓ **Ariel Publishing's *Inside BASIC* on disk**
...and other related BASIC programming information and tools.

✓ **75 MB of Special Demos relevant for developers**

FREE!
THINK™ Reference
Symantec's THINK Reference 2.0. Complete on-line guide to Inside Macintosh, Vol. I-VI, with cross referenced index, detailed information of each function, procedure and detail needed when programming the Macintosh.
SYMANTEC.™

MacTech CD-ROM, Volumes 1-10:
$49 plus shipping and handling. $39 plus shipping and handling for upgrades from any previous version of the CD.

http://www.mactech.com



(Need we say more?)

For Macintosh Programmers & Developers

MacTech™
MAGAZINE

by Stephen Humphrey, VP Engineering, Acorde Corporation

Cyberdog, the OpenDoc Internet Components

The future of Internet surfing is OpenDoc

Cyberdog is the code name for Apple Computer's OpenDoc-based Internet components. This article provides a 10,000-foot view of the Cyberdog architecture and a cursory introduction to the most important parts of its API.

Cyberdog will include components which provide Web browsing, SMTP and POP mail, Usenet News, FTP, Gopher, and Telnet. These components provide the sort of functionality one expects from Internet apps today, plus strong mutual integration with one another, and with a universal log, which keeps a historical record of the user's actions, and a notebook, which stores pointers to the user's favorite places and people. For instance, Cyberdog's integrated SMTP and POP mail system (see Figure 1) is fully MIME-capable; addresses may be stored in the notebook for easy access; and any CyberItem may be sent as an enclosure. The News reader shows the familiar display of newsgroups and messages (see Figure 2), and any icon can be dragged to the notebook. Cyberdog is designed to improve the Internet experience for MacOS users by closely integrating the various

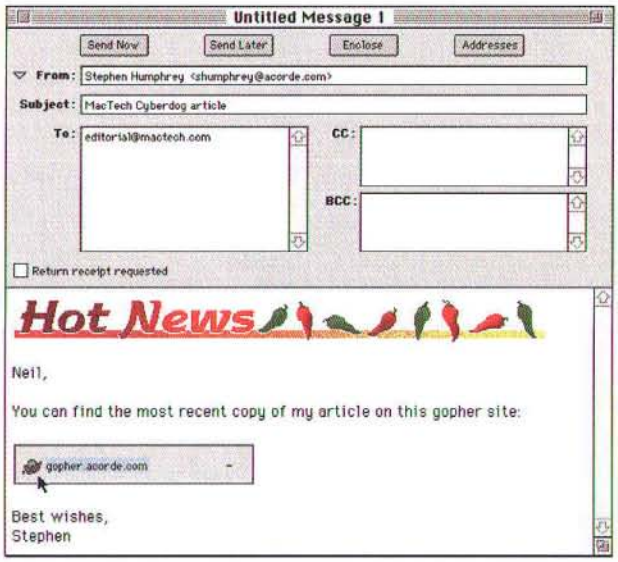


Figure 1. A Mail window

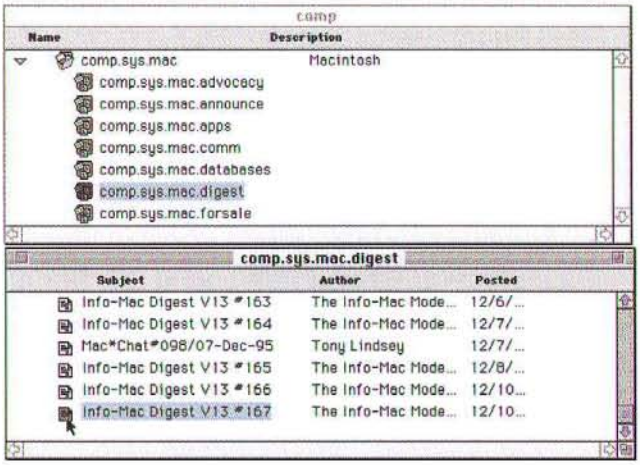


Figure 2. The News reader

Stephen Humphrey – Stephen Humphrey is Vice-President of Engineering for Acorde Corporation, an Internet-components developer. An eleven-year Macintosh veteran, he is also the author of *OpenDoc Developers Guide: Macintosh Components*, due February 1996 from Hayden Books (http://www.mcp.com/hayden/tech/). He can be reached at shumphrey@acorde.com. For more information on Cyberdog and OpenDoc, contact the famous Jim Black, OpenDoc Evangelist, at black@apple.com.

components with each other, with other applications, and with the desktop. Apple will encourage third-party developers to extend and replace the base Cyberdog components by fully documenting the Cyberdog API and architecture.

As I write this article, Apple plans to distribute a sneak preview of the shared libraries which make up Cyberdog on the OpenDoc Developer Release 4 CD [in this issue]; the end-user release of the Cyberdog components is currently scheduled for May 1996.

The Cyberdog components factor their Internet responsibilities into three major areas: Viewers, Services, and Context facilities. Viewers are responsible for displaying the myriad data types commonly found on the Net, like JPEG and HTML. Services manage the protocols used for transporting the data types. Context facilities hold the history of the user's interaction and help the users keep track of their favorite sites, newsgroups, and mail addresses.

Viewers are OpenDoc Part Editors

One of the most exciting aspects of Cyberdog is its extensive use of OpenDoc. Cyberdog depends fully on the OpenDoc architecture for displaying and interacting with Net-borne data. In fact, every part of Cyberdog that has a user interface(UI) is implemented as an OpenDoc part. Even those components which do not have a UI are implemented as SOM objects, so again they behave similarly to OpenDoc components. This dependence on OpenDoc means that a Cyber-aware Viewer you write will automatically benefit from the strengths of the OpenDoc architecture. So, for example, if you write a Cyberdog-savvy Stock Ticker, your users will be able to display dynamically-updating information about their portfolios in any OpenDoc container.

Cyberdog viewers are first and foremost OpenDoc viewers. To show a data type in Cyberdog, you first implement an editor based on `ODPart`. All of a regular OpenDoc part's methods are required, and the part editor uses the standard OpenDoc event, layout, and storage facilities. To add the functionality of Cyberdog, you add an extension to your editor which inherits from `CyberPartExtension`.

`CyberPartExtension` is a virtual class which provides the methods with which the other Cyberdog components will interact with your viewer. You will write an extension which inherits the base functionality of `CyberPartExtension` but which also knows about the particular details of interacting with your editor. So for example, if you have already written a JPEG display part using OpenDoc, you will write a CyberJPEGExtension which provides your part with the additional capabilities of retrieving the JPEG data from the Net instead of just from your OpenDoc StorageUnit. CyberPartExtension is a standard ODExtension, so you will provide your extension to Cyberdog via the standard `ODPart::HasExtension()` and `ODPart::GetExtension()` mechanisms. After your part is initialized but before you create your first display frame, Cyberdog will tell your editor to use a `CyberService` to retrieve its data.

CyberServices Encapsulate Internet Protocols

A `CyberService` is the base class which manages a single Internet protocol. The basic Cyberdog components include CyberService implementations for HTTP, FTP, Gopher, Telnet, and the local file system. Notice that CyberServices represent transport protocols and not data types, so there is an FTPService, not a JPEGService. A CyberService's most important role is as a manager of a few other classes of objects which actually implement a full Internet protocol, particularly the `CyberItem` and the `CyberStream`.

A `CyberItem` represents the address of a piece of data on the Net. In its simplest form, it can be thought of as an objected-oriented wrapper for a URL. In practice, there is nothing to stop much more advanced capabilities in a CyberItem, such as complete database queries. CyberItems are portable; they can be saved in your StorageUnit and reinstantiated later. If you are implementing a viewer which can contain one or more links to outside data, such as an HTML viewer, you will save CyberItems in your StorageUnit as part of your own content model, retrieving them when necessary based on action from the user. CyberItems have no inherent UI, so they inherit from `SOMObject` instead of `ODPart`.

A simple example of a CyberItem's behavior is shown by a CyberButton, a simple Cyber-savvy viewer which comes with Cyberdog. A CyberButton is an OpenDoc part which behaves as a button (surprise!); it can be displayed in any OpenDoc container, can display a title or a picture on its face, and can be clicked on by the user. Internally, a CyberButton holds one CyberItem. When the user clicks on the CyberButton, the button calls the CyberItem's `Open()` method. This is a fire-and-forget call; the CyberButton is not responsible for any additional interaction with the CyberItem. The CyberItem is free to take any appropriate action, the most common of which is to open a viewer to display the data pointed to by the CyberItem.

As a Cyber-savvy viewer, you become interested when the `SetCyberItem()` method of your CyberPartExtension is called. This tells you that you are being opened because the user fired a CyberItem, and you are provided a reference to that CyberItem. Your most common action then will be to ask the CyberItem to create a CyberStream through which you will get the data to display; to do this, call the CyberItem's `CreateCyberStream()` method, and it will return a CyberStream.

CyberStreams Provide Clean Data to Viewers

A CyberStream implements the actual passing of data from a protocol to a viewer. After receiving a CyberStream from a CyberItem, you will tell the CyberStream to either `Open()` or `OpenWithCallback()`. This call tells the CyberStream that it should immediately begin downloading the data. It is the CyberStream's responsibility to begin retrieving the data asynchronously and to store it until you ask for it.

If you opened the CyberStream by calling `Open()`, you will poll it for data by calling its `GetStreamStatus()`

Take a Power Trip to a New Realm of Developing

Powerful Tools to Take You Where You Want to Go

You've heard about the speed of PowerPC™ and you're thinking about the cool applications and content this amazing new chip will allow you to create. Read on to learn more about the tools that will take your development where you want to go, fast.

Prototyping HyperCard® 2.3 provides a robust prototyping and development environment for professional developers. It is also an easy-to-use tool for building a wide range of custom software applications for Macintosh®, from training simulations to information kiosks. And now it's accelerated for Power Macintosh®.

PowerPC and 68K Compilers Metrowerks CodeWarrior Gold is the industry's first native development environment for both the PowerPC and 68K-based Macintosh environments. You can easily target the entire Macintosh installed base with code created using one development environment running at Power Macintosh speeds.

Cross-Platform Multimedia Create innovative, interactive multimedia titles which will play on both Mac OS and Windows systems. Give users the ability to cruise up, down and around objects. They can even zoom in and pick them up. No wonder QuickTime™VR Authoring Tools Suite v.1.0 has won so many awards, including MacUser Magazine's "Eddy" for Breakthrough Technology of the Year.

There's a wide variety of development tools available through the Apple Developer Tools Catalog. Check it out. Developing on the Macintosh platform has never been easier or faster. And now with cross-platform flexibility, it's smarter than ever.

QuickTime VR Authoring Tools Suite v.1.0

Develop cross-platform multimedia products with QuickTime VR content

$495

This suite includes complete documentation for planning, designing, photographing and capturing content. QuickTime VR uses breakthrough compression/decompression techniques which produce small file sizes. And there's no special hardware required.

How to Order

To order call:

1 800 282-2732

And ask for a free copy of the Apple Developer Tools Catalog.

Visa, MasterCard and AMEX accepted.

Metrowerks CodeWarrior Gold

A complete development solution for C/C++ and Pascal programming.

$399

Runs native and cross-compiles on 68K or PowerPC platforms, ensuring that as you go forward on Power Macintosh you also conserve your investment in your installed code base on 68K Macintosh and your code base remains backward-compatible. Lets you compile, link, and debug 68K software at full native speed on a PowerMac™, and build PowerPC versions of your software on a 68K Mac.

HyperCard 2.3

A robust prototyping and development environment accelerated for Power Macintosh

$99

Comes with powerful scripting tools including AppleScript™ Runtime built-in, a modeless script editor, hypertext support, debugging tools, new 24-bit color painting tools, Automated Button Tasks, and many other features. Stacks can be saved as stand alone, double-clickable applications you may distribute without software royalty fees.

MT296

method. The most interesting replies are `kDataAvailable` and `kDownloadComplete`. If you opened the CyberStream by calling its `OpenWithCallback()` method, then the CyberStream will notify you whenever data is available by calling the notification method you register.

Any time the CyberStream has data available, you can request a chunk of data from the stream with the call `GetBuffer()`. When you are finished processing the data, you must call `ReleaseBuffer()`. If you are using a callback method to notify you when data is available, you must remember that this notification may happen at interrupt time; you will not be able to allocate memory, draw to the screen, or perform any other action which is not interrupt-safe. However, it is okay to set an internal state which will get and process the data later, such as at idle time. The CyberStream may only have a limited number of buffers, so it is a good idea to release them as soon as you are able to. You will continue calling `GetBuffer()` and `ReleaseBuffer()` until the CyberStream reports it is finished downloading.

CyberStreams are responsible for parsing the data stream and removing any protocol-specific headers or similar data blocks in the stream. This has the advantage of providing the viewer with a consistent stream of data regardless of the data's transfer protocol on the Net. So for example, your JPEGViewer need not care whether the CyberItem it receives is really a GopherItem, an FTPItem, or a FileItem; regardless of the protocol the user chose, the JPEG stream you receive will be the same.

One limitation of the method described above is that sometimes a CyberItem doesn't know what kind of viewer it should open. For example, a WebItem cannot open an appropriate viewer until it knows the kind of data at which it is pointing, that is, until it parses the HTML and finds an appropriate data-type tag. In this case, the CyberItem will actually open the CyberStream and start it downloading even before the real viewer is opened. The CyberItem will also open a special OpenerPart that will display the download status until the real viewer can be determined and opened. However, as a viewer you will not know or care that the stream has already been opened; you will ask the CyberItem to create a CyberStream, ask the CyberStream to open, and begin polling as usual.

As of now, CyberStreams are designed primarily to pass data in one direction. This is decidedly unhelpful for some protocols which depend on more interactive communication between the viewer and the stream. For example, the telnet protocol cannot be implemented efficiently using CyberStreams. Thus, when a TelnetItem (a telnet CyberItem) asks a TelnetViewer to open, the TelnetViewer never requests a TelnetStream. Instead, it just fully implements the telnet protocol within the viewer by asking the TelnetItem for its connection information and creating the connection itself. Since no CyberConnection object exists, this means implementing these types of protocols is fairly tedious today. This is a great opportunity for either a future version of Cyberdog or for a smart third-party.

Context Facilities Tie the Parts Together

Cyberdog provides several built-in context facilities which unite the various components into a seamless Internet workspace. These include a common Connect dialog, a Preferences panel, the Log, and Notebooks. Each of these is managed through the single `CyberSession` object.

The CyberSession is responsible for the overall integration of the Cyberdog components. It is similar in purpose to the OpenDoc session object, `ODSession`, although it is different in the particular services it provides. There is at most one CyberSession for each ODSession. The CyberSession is the main facility through which Viewers will request various Cyberdog objects. It is also the facility through which standard OpenDoc containers will be able to add the Cyberdog menus to their menu bar. Among its responsibilities, the CyberSession checks the Cyberdog libraries folder to see which CyberServices are available. This is what allows the run-time addition of new services to Cyberdog's repertoire.

CyberServices may provide a Connect panel. If provided, this part allows the Cyberdog Connect dialog to display protocol-specific fields for any service available to the user. In operation, the Connect dialog is reminiscent of the pre–System 7 Control Panel dialog, with scrolling icons on the left and individual panels on the right. Since the panels are implemented as regular OpenDoc parts, a service which

implements a new protocol can easily provide a panel for the CyberSession to display to the user. Such a Connect panel is implemented by adding a `CyberPanelExtension` to a regular part subclassed from `ODPart`.

Similarly, any CyberService can provide a Preferences panel that the CyberSession will display in the Cyberdog Preferences Dialog. A Preferences panel, too, is implemented by adding a `CyberPanelExtension` to a regular part subclassed from `ODPart`. (This implementation of the Cyberdog dialog boxes provides one of the best non-document uses of OpenDoc to date; it validates OpenDoc as more than just a compound-document architecture.)

To provide an historical context for the user's actions, Cyberdog provides a universal Log which tracks where the user has been on the Net. The user can show the log, display its items hierarchically, alphabetically, or historically, and return to places in it by simply clicking on the place's icon (see Figure 3). A Viewer posts a new item on the log by providing a CyberItem (and optionally, its hierarchical parent) to the CyberSession's `AddCyberItemToLog()` method.

Figure 3. The Log window

The user may also save one or more Cyberdog Notebooks (see Figure 4). These simple lists of CyberItems have a single-level folder system in which the user can organize favorite places and people. The user identifies a default notebook, and Viewers may add an item to this default notebook by simply telling the CyberSession to `AddCyberItemToNotebook()`. More typically, Viewers allow the user to drag CyberItems to a Notebook using the OpenDoc drag-and-drop facilities (so the user can drag e-mail addresses, newsgroups, Web sites, Gopher directories, and telnet connections right into a Notebook). Like other parts of Cyberdog, the Notebook is designed to implement the minimum functionality required by a beginning Internet user; it is ripe for replacement by an enterprising third-party.

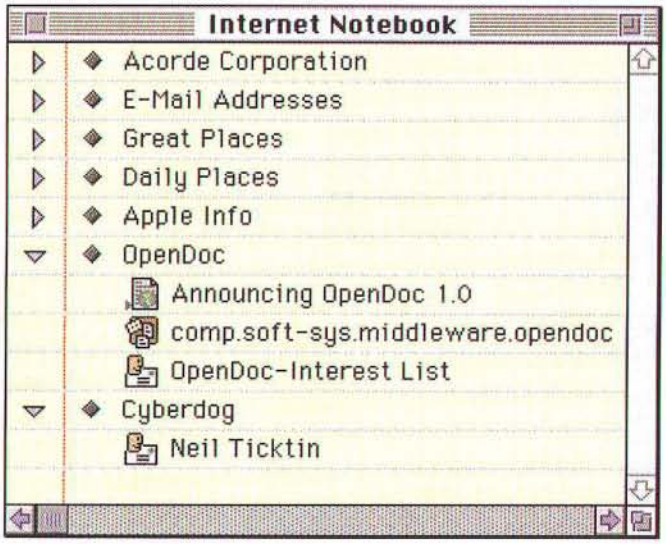


Figure 4. The Notebook

Where To Go From Here

By the time you read this article, the Cyberdog components and SDK should be available on the OpenDoc DR4 CD. The SDK is quite preliminary, but should be sufficient to get started with developing for Cyberdog (especially if you're not afraid to bleed a little). Various listservers have been established to facilitate Cyberdog discussion. Mail a message to cdog@apple.com with subject "DEV-INFO" for more information.

Apple has done a commendable job of designing Cyberdog. From its foundation, Cyberdog's architecture permits and even encourages third-party developers to replace and extend it. Since it is still alpha-quality code, expect some pain. However, developers who learn to walk the 'dog now might discover opportunities and markets that will be harder to find later. If you start early, the 'dog will probably bite you occasionally; but once you've learned to handle it, you'll be able to cuddle up close.

OPENDOC

By Gerry Kenner, University of Utah, and David Kenner, Phone Directories

Using OpenDoc With Object Flow System (OFS)

Get up and running with OpenDoc

Introduction

OK. You have taken your CD containing OpenDoc, downloaded the necessary documents, and have used PartMaker to create a generic program named HelloPart. You definitely have a warm feeling of competence because this could not have been done without doing everything precisely as the instructions called for (remember that SOM file which you failed to name in version 1 because you overlooked the instruction to do it?). What do you do now? Start reading the documentation? Heaven forbid. At least not until you have a better idea about what OpenDoc is and what it can do.

We propose to teach OpenDoc using the following steps:

1. Define OFS.
2. Explain what OpenDoc is and provide a glossary of major terms.
3. Do an overview of Apple's SamplePart (C++) project.
4. Outline a tutorial project which will create a module consisting of two parts. When completed, this project will show a menu bar with an item for selecting the name of a PICT file (ImagePart). A SFGetFile dialog box will appear (SelectPart) asking for the name. The name will then be placed into persistent storage for access by the ImagePart. The name will be retrieved by the ImagePart and displayed in its window.

OFS

OFS (Object Flow System) is an intuitive method for doing program design. As our CAD tools, we use a flow-charting program (MacFlow™, Mainstay) and a word processing program with outlining capabilities. An introduction to OFS can be found in the December, 1993 issue of *MacTech* (see Bibliography, below). Three unpublished articles dealing with the subject can be obtained upon request via email. The details of the outlining level of OFS are given in Appendix A at the end of the article.

OpenDoc

OpenDoc is a development platform which takes binary code fragments (objects) and integrates them so they will interact with each other. Basically, there are four types of code fragments involved: user, third-party, libraries and platform-specific (drivers). What is earth-shaking about the OpenDoc concept is that it does not require source-level code. Translated, this means that (1) you can program with whatever language you wish as long as the OpenDoc interface rules are followed; (2) changes can be made in OpenDoc without having to recompile the entire program; and (3) the code is portable and can be distributed in binary form.

Where to Find OpenDoc

The latest version of OpenDoc can be obtained by sending a message to opendoc@applelink.apple.com. Additional information about OpenDoc is available via the World Wide Web pages at http://www.info.apple.com/opendoc/. The home pages of the Component Integration Laboratories (CI Labs, http://

Gerry Kenner and David Kenner – Gerry Kenner is a researcher at the University of Utah, Salt Lake City, UT. His major goal is developing better methods for doing program design. David Kenner is a Macintosh programmer at Phone Directories, Orem, UT. They can be reached via email at Gerry.Kenner@m.cc.utah.edu.

www.cilabs.org) are a rich source of information about all aspects of OpenDoc. [And, there's the CD in this issue!]

OpenDoc and SOM

Figure 1 shows an oversimplified view of the relationship between OpenDoc and the other components of the Macintosh system. The figure was derived empirically and is probably not completely accurate, but it is useful as a working model. Note that "SOM stubs" refers to the SOM interface.

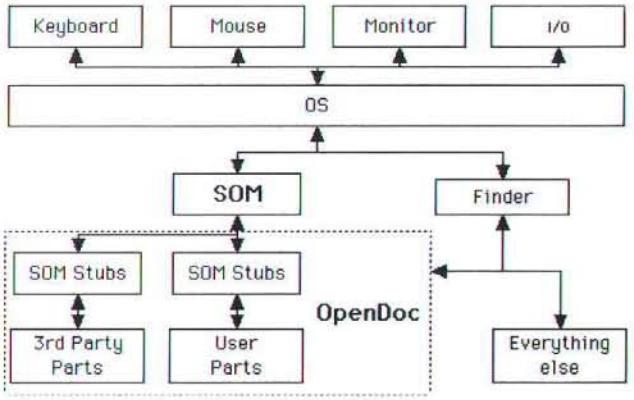


Figure 1. OpenDoc, SOM and user-created parts.

Glossary

Here are some terms you will need to know to understand the material which follows.

OpenDoc – A development platform designed to seamlessly coordinate binary code fragments into larger entities corresponding to present day applications. Although OpenDoc is serviceable as a name, a functional name such as Object Manager or Object Integrator would have been better.

Document – Collection of parts or objects. Basic units used by OpenDoc.

Part – Corresponds to the term "object". Consists of content (definition) and part editor (executable code).

Content – Also "part content". The portion of a part which describes its data, i.e. the object definition.

Part editor – Executable code of a part or object.

Frame – Total virtual area which can be used to display a part. A facet is displayed in some or all of the frame.

Facet – Total actual area which can be used to display a part. A facet corresponds to a portion of a window or printing canvas where a part is expected to image itself.

Focus – Designation of ownership. A focused window is one which is selected and ready to be drawn in.

Storage unit – An object for storing persistent data, i.e., data which is shared between parts.

Draft – Versions of a document maintained with incremental deltas.

Module – A group of parts.

SAMPLEPART

We will now look at how an OpenDoc part is put together and executed. We will do this by looking at the contents of the SamplePart project, followed by some diagrams which show how the different components of an OpenDoc project interact with each other.

SamplePart Project Window

Figure 2 shows the Symantec project window for the SamplePart part. What are all these files? The System libraries are the PowerPC equivalents of the MacTraps, ANSI, Sane, etc. libraries of the 68k Macintoshes. The OpenDoc libraries and OpenDoc utilities provide code which OpenDoc needs in order to operate. We can safely ignore these at this time. The resource directory contains the resources used by the SamplePart part. These include the About SamplePart dialog box, and some strings and information required for external accessing of the part and its storage. This leaves the Sample Part and SOM interface directories.

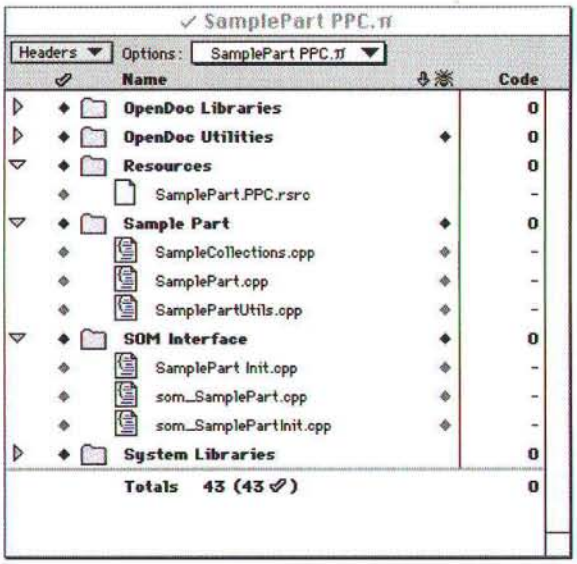


Figure 2. Symantec C++ project window for SamplePart.

The code which is of concern to us in this article are `som_SamplePart.cpp`, located in the SOM Interface directory, and `SamplePart.cpp`, in the Sample Part directory. Examination of the contents of the `som_SamplePart.cpp` file shows that it consists mainly of interface or stub calls to the corresponding C++ methods in the `SamplePart.cpp` file.

Reference to Figure 3 shows how this works. The SOM interfaces are the clearing points for receiving and sending messages to SOM. This is done by programming in the IDL (Interface Definition Language). The programmer can write the entire part in IDL, or alternatively, write interface code which relays the SOM messages to code written in higher level languages, such as C++. Figure 4 shows an example of how this works.

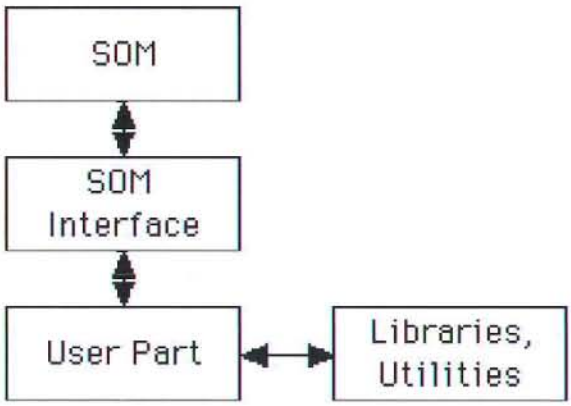


Figure 3. Closer look at SOM relationships.

In this case, SOM sends a message to the SOM interface requesting that the SamplePart part be instantiated. Thus:

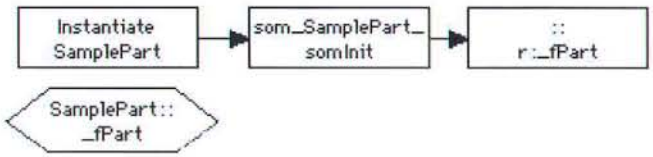


Figure 4. Program flow for instantiating SamplePart.

Other files in the SOM interface and the Sample part include `SampleCollections.cpp`, `SamplePartUtils.cpp`, `SamplePart Init.cpp` and `som_SamplePartInit.cpp`. `SampleCollections.cpp` and `SamplePartUtils.cpp` contain functions and classes with methods for doing things such as copying strings and making conversions between global and local coordinate systems. These classes do not interact directly with the SOM interface code. As their names imply, `som_SamplePartInit.cpp` and `SamplePart Init.cpp` contain some short initialization sequences.

Startup Code

Let's take a closer look at what happens when an OpenDoc part is started up. This is illustrated in Figure 5. The first column denotes messages received from SOM. The second column denotes the instantiation of an object of SOM class `som_SamplePart` and shows execution of some of its methods. Similarly, the right-hand column shows the instantiation of an object of the C++ class `SamplePart` and the execution of many of its methods. The diagram has been simplified by omitting some messages.

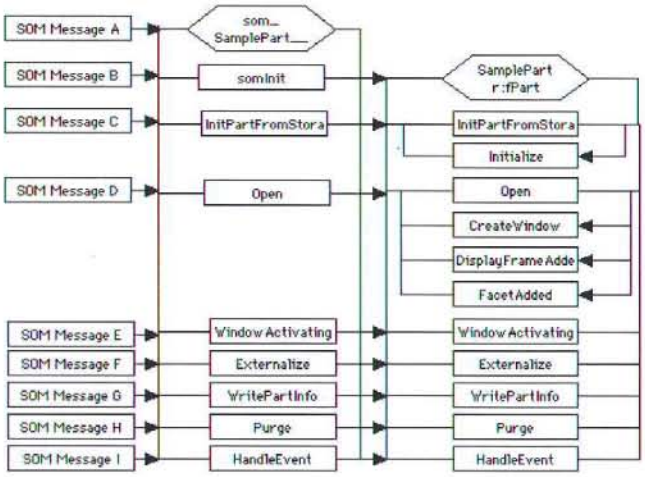


Figure 5. Start-up sequence for SamplePart.

Except where method calls are made from within the SamplePart object (e.g. `Initialize` is called by `InitPartFromStorage`), all activity proceeds from left to right. For example, the first message received results in the instantiation of an object of the SOM class `som_SamplePart`. The second message then calls the `somInit` method of the `som_SamplePart` class which instantiates an object of the C++ class `SamplePart`. Calling of constructor methods is not shown as it is assumed this happens when an object is instantiated.

Everything is pretty straightforward here. The steps for creating and displaying a part are opening it followed by creating a window. Space for displaying the frame is added, after which a facet is created. The part is then fine-tuned and a copy is sent out to memory (`Externalize`). The program then goes into idle (`HandleEvent`). The screen is drawn as an update event during idle (Figure 6).

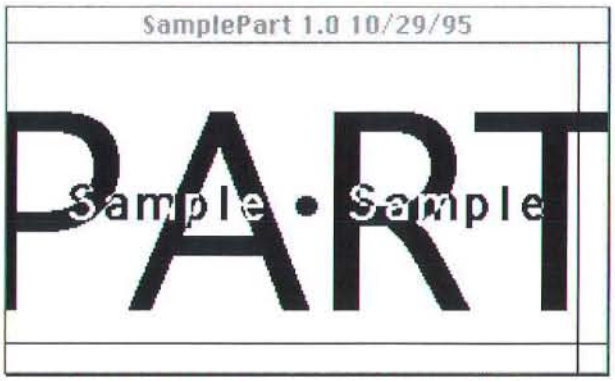


Figure 6. The SamplePart display.

Shutdown Code

Figure 7 shows the operations which occur when the SamplePart part is shut down. Note that the objects of classes `som_SamplePart` and `SamplePart` were instantiated at startup and the process is not repeated.

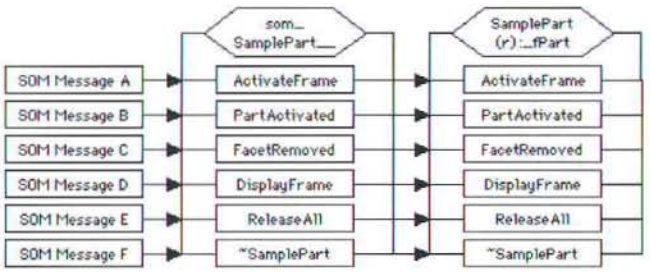


Figure 7. Shutdown operations (simplified for clarity).

In this case, the window (not shown), frame and part are all activated, and the facet is removed. Storage is then released and the destructor method is called.

Tutorial

Our next step is to design and create a module named **KSS** using OFS. The module will have two parts named `ImagePart` and `SelectPart` respectively. This is the first step towards creating a full-fledged image analysis program for measuring surfaces and areas. Hence the name of the `ImagePart` part, which will eventually become the container for the completed project.

Intuitive logic plus a general perusal of program design books indicates that there are approximately four levels of programming. These are:

- Planning level
- Prototyping level
- Flow diagramming level
- Programming level

Using terminology borrowed from Goldstein and Alger (see Bibliography), we break the planning level into two further divisions, the reference and solution sublevels. See Figure 8; the border shadow denotes the presence of subcharts.

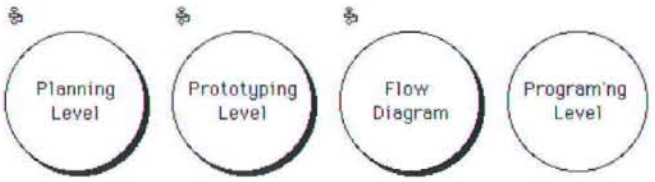


Figure 8. The uppermost level of the OFS system.

Planning Level

Reference sublevel. The template for building the ImagePart and SelectPart parts is the SamplePart part of the Symantec C++, version 8 demo projects.

Solution sublevel. The ImagePart part will be placed in the Apple container part or else run as a standalone part. The frame of the ImagePart part will contain the words "Hello World". ImagePart will add an additional menu to the menu bar containing a single item, **Open Pict**. When the menu item **Open Pict** is selected, the SelectPart part will be instantiated. The SelectPart part will display a SFGetFile dialog box from which the user will select the name of a file. The name of the file will be stored in SelectPart's persistent storage. Program execution will then return to the ImagePart part which will retrieve the name from SelectPart's persistent storage after which it will dispose of SelectPart.

The ImagePart part will then display the name of the selected file in its frame.

Prototyping Level

The prototyping level is where the project begins to get some substance. In practice, prototyping is done in many ways. These range from the use of prototyping programs such as Marksman™, AppBuilder™, Visual Architect™ and Rational Rose™ through various CAD-type systems such as the one we use here, to miscellaneous notes and ideas kept on napkins, scraps of paper or the brain of the program designer.

The root chart of the prototyping level is shown in Figure 9. It has separate boxes for the two parts of the module. The menu bar, display and diagram categories of the ImagePart and the diagram category of the SelectPart are shown in Figures 10 to 13.

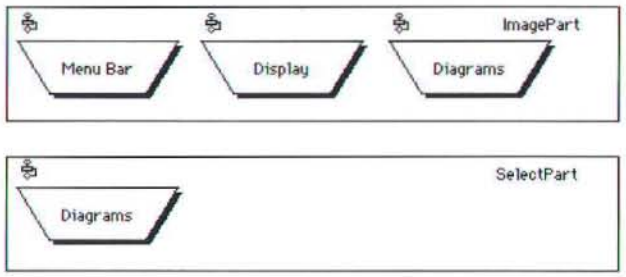


Figure 9. Root display of the prototyping level.

Figures 10 and 11 provide general ideas of what the menu bar and the screen display will look like. The layout of the menu bar is particularly important because much of the program's flow is determined by the layout. In a full-scale part, there would also be diagrams showing the layout of various windows and dialog boxes.

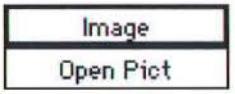


Figure 10. Proposed menu bar add-on.

Figure 11. Proposed display.

Figure 12 shows the proposed program flow for the ImagePart when the **Select File** item of the **Image** menu is selected. Note the instantiation of the SelectPart part.

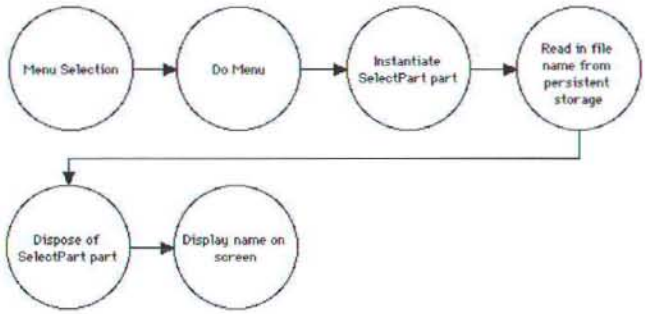


Figure 12. Program flow for ImagePart.

When the SelectPart part is instantiated, it is initialized, followed by the sequence of events shown in Figure 13.

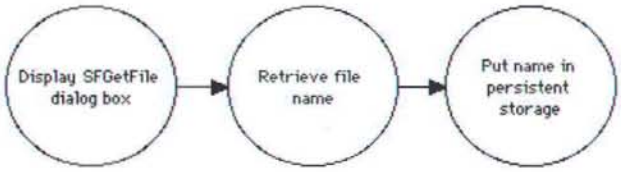


Figure 13. Program flow for SelectPart.

Flow Diagramming Level

At this point, it is time to start laying out how the module is to be implemented. This is where we take our existing and third-party code and combine it with a knowledge of what types of programming we know how to do, and lay the program out in detail. When we get done, we should have all our classes and methods identified and how they interact with each other.

Figure 14 shows how ImagePart will be implemented. The main points of interest here are that the `MyOpenPict` method is called from `HandleEvent`, and the use of operations A (`CreatePart`) and B (`GetStorageUnit`) where SelectPart is instantiated and the file name is retrieved from persistent storage.

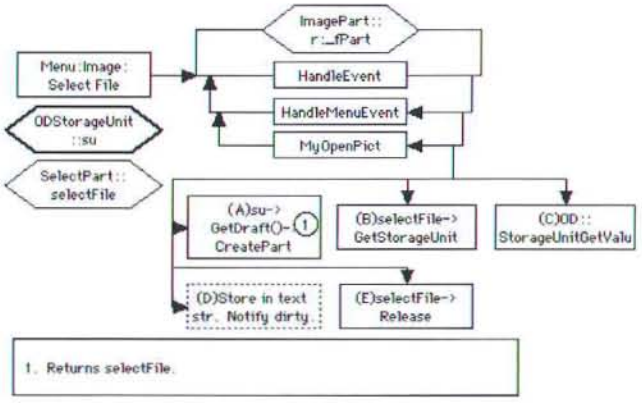


Figure 14. Flowchart for ImagePart.

Figure 15 shows that the `GetFileName` method is called from the `InitPartFromStorage` method.

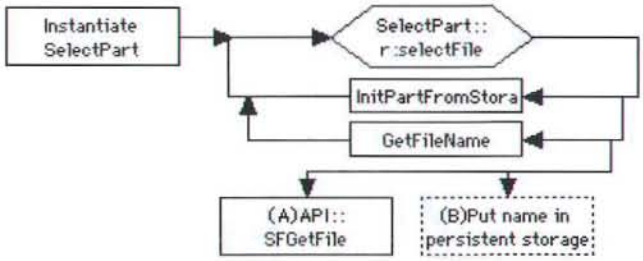


Figure 15. Flowchart for SelectPart.

Programming Level

This is where the actual code is written. Ordinarily, this section is empty because the programmer would refer directly to the project files for information. This is particularly the case since the browsers and editors which are provided with the compilers, greatly simplify the task of finding one's way around the code.

In this case we have made an exception because we are demonstrating how to set up a simple program showing how to use multi-parts as well as accessing persistent storage. As was done above, the presentation is made in two parts, one each for ImagePart and SelectPart.

IMAGEPART

Build the project using PartMaker or the procedure given in Appendix B. Use **ImagePart** and **KSS** as the class and module identifiers, respectively.

First we need to modify the header declarations. Add the following lines to the end of the `ImagePart.h` file. Put these lines at the end of the methods declaration:

```
// • User methods •
void   MyOpenPict(Environment *ev);
```

Put this line in the private variable declarations:

```
Str255      fTextData;
```

Now we come to displaying a message on the screen. Add the following lines to the end of the Initialize method

```
strcpy((char*)fTextData, "Hello World!");
c2pstr((char*)fTextData);
```

The screen is redrawn in response to update events during idling. The code for doing this is located in the `FrameDrawView` method.

In the `FrameDrawView` method, remove the code after

```
frameWidth = (**frameRgn).rgnBBox.right -
    (**frameRgn).rgnBBox.left;
```

and replace it with the following:

```
ODSLong rfRef;
{
  CUsingLibraryResources fil;
  PenState penState;
  GetPenState(&penState);
```

```
    PenNormal();
    ShowPen();

    Rect   rct;
    short  x, y;
    rct = (**frameRgn).rgnBBox;
    ::FrameRoundRect(&rct, 40, 40);
    x = (rct.right - rct.left -
        ::StringWidth(fTextData)) / 2;
    y = (rct.bottom - rct.top - 12) / 2;
    ::MoveTo(x, y);
    ::DrawString(fTextData);
    SetPenState(&penState);
  }
  frameShape->Release(ev);
```

Next, we need to implement our menu item. The following changes are necessary in order to be able to choose the **Open Pict File** item of the **Image** menu.

Open the `ImagePartOtherResources.rsrc` file with ResEdit and add the **Image** menu (resource item **5000**) with a single item named **Open Pict File** (see Figure 10).

Add the following items to the `ImagePartDef.h` file.

```
// The "Picture" menu's resource and menu ID:
#define kImagePartMenuID   5000

// Menu item IDs.
#define kOpenPictItem    1

// Menu command numbers. Must start at 20000
#define kOpenPictCmd   20000

// SelectPart items
#define kSelectPartKind
        "Apple:Kind:SelectPart"

// Storage type
const  ODPropertyName  kPropSelectPartName =
        "SelectPart:Property:Name";
```

`kSelectPartKind` is an identifier for SelectPart so that it can be found and used.

Add the following items to the end of the `Initialize` method. Be sure `gMenuBar = session -> GetWindowState(ev) -> CopyBaseMenuBar(ev)` is called before this code.

```
  {
    CUsingLibraryResources fil;
    fMenu = ::GetMenu(kImagePartMenuID);
    if (fMenu)
      ::DetachResource((Handle)fMenu);
  }
  if (!fMenu)
    DebugStr("\pGetMenu failed");
  gMenuBar->AddMenuLast(ev, kImagePartMenuID, fMenu,
        fSelf);
  {
    CUsingLibraryResources fil;
    gMenuBar->RegisterCommand(ev, kOpenPictCmd,
        kImagePartMenuID, kOpenPictItem);
  }
```

Add to the HandleMenuEvent method:

```
case kOpenPictCmd:
  MyOpenPict(ev);
  break;
```

Now we add the MyOpenPict method. It will have the following code:

```
void ImagePart::MyOpenPict(Environment *ev)
{
  ODPart   *selectFile;
  ODStorageUnit *su;
  ODStorageUnit *pbSU;
  Str255   str;
  unsigned long size;

  su = fSelf->GetStorageUnit(ev);
  selectFile = su->GetDraft(ev)->CreatePart(ev,
      kSelectPartKind, kODNULL);
  if (selectFile != kODNULL)
  {
    pbSU = selectFile->GetStorageUnit(ev);
    pbSU->Focus(ev, kPropSelectPartName,
      kODPosSame, kODISOStr, 1, kODPosFirstSib);
    size = pbSU->GetSize(ev);
    StorageUnitGetValue(pbSU, ev, size, str);
    strcpy((char*)fTextData, (char*)str);
    c2pstr((char*)fTextData);
    // Notify program that window is invalid.
  }
  else
    DebugStr("\pCannot Create Select Part");
  selectFile->Release(ev);
}
```

Now we come to the matter of creating and disposing of a part and accessing its persistent data. To create a new part you must first get a reference pointer to your storage unit. The new part is instantiated and initialized by calling the `GetDraft` method of the storage unit which in turn calls the `CreatePart` method. These calls return a reference pointer to the part.

An external part's persistent data is accessed by getting an object reference to it by calling the external part's `GetStorageUnit` method. The storage unit's focus method is then used to pinpoint the desired data. `GetSize` is used when the size of the data is unknown. Finally, the data is retrieved by calling the OpenDoc `GetStorageUnit` method.

Create a new project named SelectPart using PartMaker. Use the identifiers **SelectPart** and **KSS**. Name the folder `SelectPart`.

First, the header declaration. Add the following line to `SelectPart.h`.

```
void GetFileName(Environment *ev);
```

Now we set up the persistent variable. Add the following three lines to the end of the Initialize method.

```
ODStorageUnit *storageUnit;
storageUnit = fSelf->GetStorageUnit(ev);
storageUnit = AddProperty
    (ev, kPropSelectPartName)->
        AddValue(ev, kODISOStr);
```

The `AddProperty()->AddValue()` complex reserves space in the parts storage unit for a `kODISOStr` of the `kPropSelectName` type. `kPropSelectName` is defined arbitrarily by the programmer in the `SelectDef.h` file as:

```
const ODPropertyName kPropSelectName =
    "Select:Property:Name";
```

where `Select` is the part identifier and `Property` is an Apple term which is defined below. The term `Name` was selected by the programmer.

It is time for a few more definitions. The term "Property" is defined in the OpenDoc Class Reference as an element of a storage unit. It defines a kind of information, such as a name or a piece of data, and contains one or more values (data streams containing one or more bytes). In the case given above, the term "Property" is used as follows:

```
Part identifier:Property:User
   or Apple defined selected name of identifier.
```

Note how the term `kPropSelectName` is put together based upon its definition.

It is strongly recommended that the programmer use the above conventions, as they facilitate identifications of tokens and reduce the chances of creating duplicate names.

Another term which it would be helpful to learn at this time is `kODISOStr`. This is a null-terminated string made up of 7-bit ascii characters. Note that it is a subset of a C string.

Observe that it is crucial that the proper CI Labs ISOString prefix be used for all ISOStrings (such as `kSelectPartKind`).

This completes the changes to the `MyCommonInitPart` method. Next, we need to deal with the method `GetFileName`; add the following line to the ends of the `InitPart` and `InitPartFromStorage` methods.

```
this->GetFileName(ev);
```

Add the code for the `GetFileName` method to the end of the `SelectPart.cp` file.

```
void SelectPart::GetFileName(Environment* ev)
{
   Point          where;
   SFTypeList     typeList;
   SFReply        reply;
   int            vRefNum;
   unsigned long  len;
   Str255         str;
   ODStorageUnit  storageUnit;

   storageUnit = fSelf->GetStorageUnit(ev);

   where.h = 40;
   where.v = 40;
   {
      CUsingLibraryResources fil;
      {
         ::SetCursor(&ODQDGlobals.arrow);
         ::SFGetFile(where, 0L, 0L, -1, typeList, 0L,
             &reply);
      }
   }
   if ((reply.good == true) && (fDirty != kODFalse))
   {
      vRefNum = reply.vRefNum;
      p2cstr(reply.fName);
      strcpy((char*)str, (const char*)reply.fName);
      len = strlen((char*)str);
      storageUnit->Focus(ev, kPropSelectPartName,
        kODPosSame, kODISOStr, 1, kODPosFirstSib);
      StorageUnitSetValue(storageUnit, ev, len,
        (ODValue)&str);
      ODSUForceFocus(ev, storageUnit,
          kPropSelectPartName, kODISOStr);
   }
}
```

Now for the matter of storing persistent data. There are two steps in storing persistent data. First, space must be allocated. As shown above in the code for the `Initialize` method, this is done by getting an object reference to the part's storage unit and then calling its `AddProperty` method with name and size information.

An example of the second step is shown in the code for `GetFileName`. First, you get an object reference to the storage unit and focus on it by providing name, type and position information for the variable. Next, the OpenDoc method `SetTheValue` is called and the value is forced into storage by calling the `ODSUForceFocus` method.

Where To Go From Here

We had originally intended to show how global parameters could be passed from the ImagePart to the SelectPart. Specifically, we wanted to pass in the information that only the names of PICT files were to appear in the dialog box. Unfortunately, we ran out of time and were unable to do this. This should be a good exercise for the reader.

The reader has no doubt noticed that we put a comment into the `MyOpenPict` method saying, in effect, "Let OpenDoc know the screen needs to be refreshed." Implementing this instruction would result in an immediate screen refresh. As it stands, you must do something like put a Microsoft Word screen on top of the window and then switch back to OpenDoc to get the screen refreshed.

The next logical thing to do with ImagePart is to make it a container for other parts.

Finally, figure out how to deallocate the persistent memory used to store the file name.

Further Reading

The serious acolyte will want to read the following articles.

Alfke, J.P., "Learning to Love SOM," *MacTech Magazine*, 11:1 (1995) 12–17.

Alfke, J.P., and J. Mattson, "Opening Up OpenDoc," *MacTech Magazine*, 11:1 (1995) 52–65.

Apple Computer, "OpenDoc Cookbook for the Macintosh," OpenDoc Developer CD #3 and Apple Developer CDs.

Apple Computer, "OpenDoc Programmer's Guide for the Mac OS," OpenDoc Developer CD #3 and Apple Developer CDs.

Campagnoni, F.R., "IBM's System Object Model," *Dr. Dobb's Journal* ("Special Report"), Winter 1994/1995, 24–29.

Curbow, D., and E. Dystra–Erickson, "The OpenDoc User Experience," *Develop*, 22 (1995) 83–93.

Goldstein, N., and J. Alger, *Developing Object-Oriented Software for the Macintosh*, Addison Wesley, 1992.

Kenner, G., and D. Kenner, *Object Flow System (OFS) for Visual C++*, unpublished manuscript, 1995. (Request via email).

Kenner, G., and D. Kenner, "Outlining the Art Class Tools Menu," *MacTech Magazine*, 9:12 (1993) 56–63.

Lloyd, J., "The OpenDoc Development Framework," *MacTech Magazine*, 11:11 (1995) 35–57.

Piersol, K., "Building an OpenDoc Part Handler," *Develop*, 19 (1994) 6–16. Though outdated, this is still the best introductory article on the subject.

Piersol, K., "Getting Started with OpenDoc Graphics," *Develop*, 21 (1995) 5–22.

Rush, J., "OpenDoc," *Dr. Dobb's Journal* ("Special Report"), Winter 1994/1995, 30–35.

Appendix A: OFS Programming at the Flow Diagram Level

An example of the topmost chart of the flow diagram level is shown in Figure A1. The symbols contain shortened names of subprojects. Other names for subprojects are **subjects** or **program components**. Each subproject is linked to a subchart containing names of component subprojects or a flow diagram showing program execution flow. The flow diagram is the lowest level.

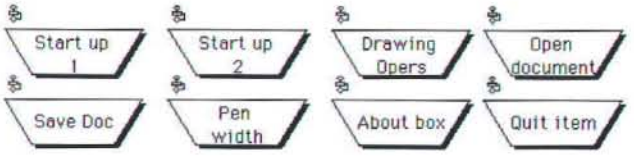


Figure A1. Example of the topmost chart of flow diagram level.

There are two types of flow-charting, object flow and method flow. Figure A2 shows the bare notation of object flow diagramming.

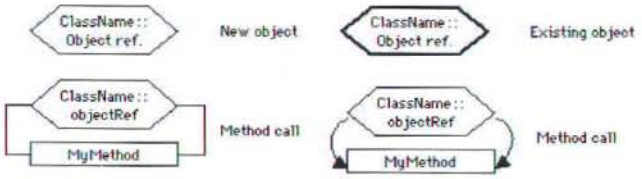


Figure A2. Object level notation: objects and methods.

The object flow level is based on the use of a hexagonal symbol containing the name of the class of which the object is an instance, and the name of the object reference, if used. If the symbol has a plain border then the object was instantiated somewhere on the page which contains the symbol. If the symbol has a heavy border, then the object was instantiated elsewhere and is being used on the present page. The bottom part of Figure A2 shows how a method call is symbolized. Note that there are no directional arrows. These were deleted to simplify diagramming.

Figure A3 shows how to execute some common operations at the object flow level. Several examples of how to do this are given in the main text of the article.

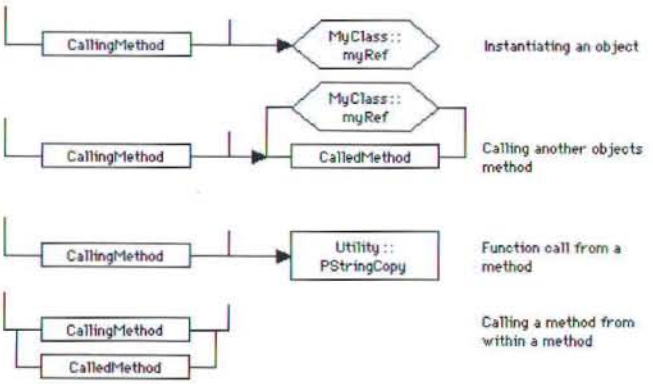


Figure A3. Object level notation: common operations.

Figure A4 shows examples of the numbering system used to keep track of the order in which operations occur. Capitalized roman numerals are not used because they take up too much space. The hierarchy is: capitalized arabic letters, arabic numbers, lowercase arabic letters, lowercase roman numerals. The sequence is repeated if more sublevels are needed.

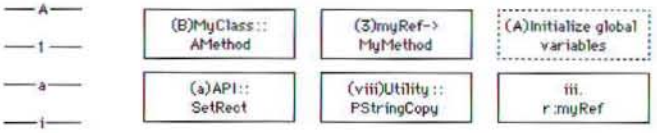


Figure A4. Object level notation: hierarchy.

In our system, method flow-charting is the lowest level before coding begins (Figure A5). At this level, operations are shown in sequence. The screen is divided into two unequal parts. The left-most quarter holds the execution start points and the emblems containing class names and object references. The right-most three-quarters contains the programming operations. A comments box is usually present at the bottom.

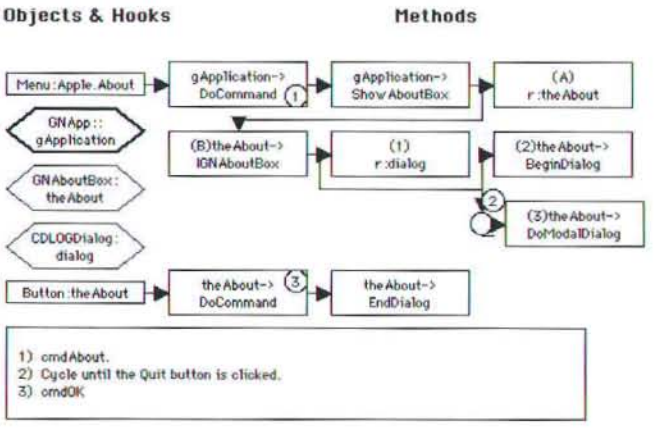


Figure A5. Method level flow-charting example.

Figure A6 shows the symbolization used in method flow-charting. Most of these are self explanatory.

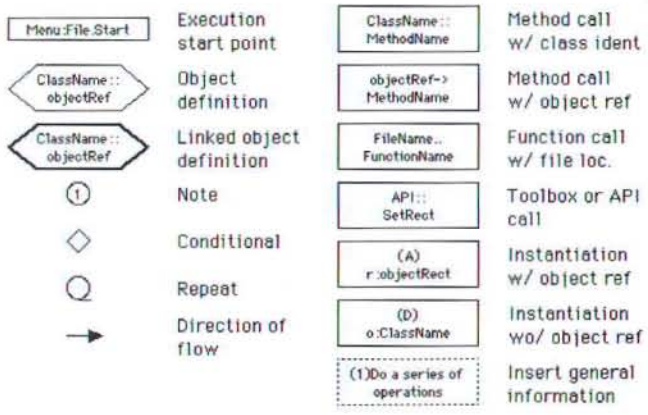


Figure A6. Symbols used by OFS for method level flow-charting.

APPENDIX B: CONVERTING THE SAMPLEPART TEMPLATE

This is how to create a project folder named `ImagePart`. It requires about thirty minutes to do a conversion. Make copies of the `SamplePart` and `Build Support` folders onto your hard drive. The `Build Support` folder does not need any changes. Using a good editor such as BBEdit™ or BBEditLite™ (BareBones Software), find and replace all instances of the following words in the files of the `SamplePart` folder.

`SampleCode`	Change to your company identifier. In our case, this is **KSS**.
`SamplePart`	Change to the new class name. In this case, **ImagePart**. **Image** or any other name will also work as long as it is distinctive.
`SampleCollections`	Change to **ImageCollections**. Case is not important.

Close the editor and go to the `SamplePart` folder. Change all instances of the word `Sample` to **Image** in the names of the folders and files. For instance, change the name of the main folder from `SamplePart (C++)` to **ImagePart (C++)**.

Open the resource file `ImagePartOtherResources.rsrc` and remove the `'vers'` resource. It causes a conflict when the project resource is rebuilt.

Using SARez (from Pascal compiler folder) or Rez, recompile the project resource. The source file is `ImagePart.r` in the source folder and the target file is `ImagePart.PPC.rsrc` located in the `Object:PPC:` directory. Use replace to create a new `ImagePart.PPC.rsrc` file. The pathway to the headers files is `OpenDoc:Interfaces:Rez:`. Load all the files.

Open the `ImagePart` project file and replace all instances of sample files with the renamed Image files.

Build the project.

OpenDoc Developer Release 4

IMPORTANT – READ CAREFULLY BEFORE USING THE SOFTWARE

Apple Computer, Inc. Software License

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT FOR A REFUND.

1. License. The software accompanying this License (the "Apple Software") is licensed, not sold, to you by Apple Computer, Inc. ("Apple"). Apple and/or Apple's licensor(s) retain title to the Apple Software. The Apple Software accompanying this License and any copies which this License authorizes you to make are subject to this License.

2. Permitted Uses and Restrictions. You may use and install the Apple Software on as many of your Apple-labeled or Apple-licensed computers ("MacOS computers") as are reasonably necessary to develop OpenDoc-compatible parts, containers, and applications designed to operate in combination with MacOS computers ("Applications"). You may not use the Apple Software for any other purpose, including but not limited to, in the development of, or for incorporation into, operating system software. In connection with the development of your Applications, you may also use, incorporate into your own Applications, compile, copy and distribute (in object code form only) as part of your Applications, the Apple Software identified in the Licensing Information folder located on the top level of this CD, provided you reproduce on each copy the copyright information contained on the original copy of the Apple Software. Except as expressly permitted in this License, you may not decompile, reverse engineer, disassemble, modify, rent, lease, loan, sublicense, distribute or create derivative works based upon the Apple Software in whole or part. Your rights under this License will terminate automatically without notice from Apple if you fail to comply with any term(s) of this License.

3. Disclaimer Of Warranty. Some of the Apple Software may be designated as alpha, beta, development, pre-release, untested, or not fully tested versions of the Apple Software. Such Apple Software may contain errors that could cause failures or loss of data, and may be incomplete or contain inaccuracies. You expressly acknowledge and agree that use of the Apple Software is at your sole risk. The Apple Software is provided "AS IS" and without warranty of any kind and Apple and Apple's licensor(s) (for the purposes of Sections 3 and 4, Apple and Apple's licensor(s) shall be collectively referred to as "Apple") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. APPLE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE APPLE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE APPLE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE APPLE SOFTWARE WILL BE CORRECTED. FURTHERMORE, APPLE DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE OR IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE APPLE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. THE LICENSE FEES FOR THE APPLE SOFTWARE REFLECT THIS ALLOCATION OF RISK. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

4. Limitation Of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL APPLE BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE APPLE SOFTWARE, EVEN IF APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THIS LIMITATION MAY NOT APPLY TO YOU.
In no event shall Apple's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid for this License.

5. Export Law Assurances. You agree that the Apple Software will not be exported outside the United States except as authorized by United States law. You also agree that Apple Software that has been rightfully obtained outside of the United States shall not be re-exported except as authorized by the laws of the United States and of the jurisdiction in which the Apple Software was obtained.

6. Government End Users. If the Apple Software is supplied to the Department of Defense ("DoD") of the United States Government, the Apple Software is classified as "Commercial Computer Software" and the DoD only acquires "restricted rights" as defined in Clause 252.227-7013(c)(1) of DFARS. If the Apple Software is supplied to any other unit of the United States Government, the Government's rights are as defined in Clause 52.227-19(c)(2) of FAR or, in the case of NASA, as defined in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

7. Controlling Law and Severability. This License shall be governed by the laws of the United States and the State of California. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect.

8. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Apple Software and supersedes all prior or contemporaneous understandings regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by Apple.

FROM THE FACTORY FLOOR

By Dave Mark

A monthly column of assorted news, interviews, and technical information from Metrowerks.

In last month's issue, we introduced a brand new column with little in the way of explanation. Here's the skinny. At Neil's persistent urging, the folks at Metrowerks asked me to put together a regular monthly column, but with no particular agenda. For example, last month's column was a Java interview with Greg Galanos, Metrowerks' President and CEO. This month, we'll go through a pile of Metrowerks tech support questions and answers. Got any ideas? Any interviews you'd like to see? As always, your feedback is most welcome. Check out page 2 of the magazine for contact information.

The questions were provided by Stephen Chong, Khurram Queresbi, and the folks at Metrowerks tech support. I did a little bit of editing just to clean up the questions but I tried to keep with the spirit of the original question. Since not everyone wants their name up in lights, I didn't include names with the questions.

Top Ten Tech Support Questions

Q: My program makes extensive use of SIOUX for console i/o, and I frequently generate more than 32K worth of output in the console window. I've noticed that when I scroll down to the bottom of my console window, I occasionally end up with garbage in the window and sometimes the window stops scrolling. Any ideas?

A: Our SIOUX output window can only handle 32k of output at a time, and after you send it more than that, results are unpredictable. The solution is to either redirect `stdout` to a file (via the `ccommand()` function/dialog in `console.h`) or change the `printfs` to `fprintfs` and write to a file.

Q: How can I use the debugger for debugging MPW tools and how can I specify command-line arguments when I am debugging?

A: Currently, our debugger doesn't support debugging MPW tools. One option is to build your tool as an application that uses the `ccommand()` function to take its command-line arguments and I/O redirection. Once it is debugged, you would change the project type back to MPW tool, swap ANSI libraries, and remove the `ccommand` call. Another option is to purchase Steve Jasik's The Debugger, which can debug 68K MPW tools, and possibly PPC ones.

Q: In the following code snippet, the scope of the variable `i` inside the for-loop doesn't conform to the ARM when I compile using the CodeWarrior C++ compiler. Why is that?

```
void scopeOfVars()
{
  long a = 0;
  if (a)
    for (long i = 0; i < 12; ++i)
      a = i;
  else
    for (long i = 0; i < 12; ++i)
      a = i + 1;
}
```

A: The scope of the index is just within the for-loop; this agrees with the draft ANSI Standard for C++ which is what CodeWarrior follows. If you instead want to force ARM conformance, which allows the index to live outside the for-loop, you can do this by checking the **ARM Conformance** checkbox in the C/C++ Language Preferences panel.

Q: I have two source code files I am linking together. One is written in C and one in Pascal. Here's the Pascal source code, from source file `Foo.p`:

```
unit Foo;
interface

var
  myGlobalVariable : Integer;

implementation

end.
```

Here's the C source code:

```
extern short myGlobalVariable;

void main(void)
{
      myGlobalVariable++;
}
```

When I compile and link these files using CodeWarrior I get a linker error complaining that `myGlobalVariable` referenced from `main` is undefined. What gives?

A: You will need to do either (but not both) of the following to make your code link:

- In the Pascal source, enclose the variable declaration with the compiler directive `{$J+}` and `{$J-}`. The `$J` directive controls the case conversion of global identifiers when building object files.

or

- Use all uppercase in the variable name in your C source code.

Q: I just upgraded to CW7 and I'm having problems getting a CW6 Pascal 68K project to link under CW7. When I recompile my code, I got the following linker errors:

```
Link Error : StrOp.c 'memchr' referenced
  from '__POSITION__' is undefined.
Link Error : MWP.Stub.lib: '%_X2STR' referenced
  from 'NUM2STR' is undefined.
Link Error : MWP.Stub.lib: 'STR2DEC' referenced
  from 'STR2NUM' is undefined.
```

A: Under CW7, the IDE is now integrated, allowing Pascal and C to use the same set of ANSI C libraries. You'll need to make sure these libraries have been added to your project. To find out all the libraries needed for a typical 68k project in CW7, you might want to create a new project using the `MacOS 68k Pascal.μ` project stationery, then compare your new project to your old project.

Q: In CW6, I used the libraries `P/ANS.68K.lib` and `SetLib.Lib (A5)`. What are the CW7 equivalents?

A: Neither of these libraries are needed under CW7.

Q: I have a simple ANSI C console-based program I wrote on the Mac and that I am trying to get working under Windows '95. The program works just fine under MacOS but I can't get it to build using the Win32/x86 environment. I am using the Win32s libraries as used in the CW7 Win32/x86 tutorial but I can't get my project to link sucessfully.

A: Inside the `(Project Stationery)` folder is a folder called `Additional Project Stationery`. Drag the Win32 Console application stationery from there into the `(Project Stationery)` folder. Next, create a new project using the Win32 Console app stationery. The binary created from there should run without problems under Windows95. I just tried it with `Hello World` and it ran fine on my Win95 machine.

Q: Is there a way to "Import" the template I made in version 1 of Constructor into version 2 of Constructor?

A: Unfortunately, Constructor 1 and 2 are completely different programs (literally), and they use a different mechanism for custom types. Right now it isn't possible to import 1.0.1 templates into 2.0.

Q: How can I get a SIOUX-based program to quit without pausing when the program ends or without waiting for the user to select **Quit** from the **File** menu?

A: Try this: `#include` the file `<SIOUX.h>`, then add the following code at the beginning of `main()`:

```
SIOUXSettings.autocloseonquit = true;
SIOUXSettings.asktosaveonclose = false;
```

Q: I'm trying to debug a code resource. However, after I set a breakpoint at the beginning of the resource and then run the application that calls this resource, I never drop into the debugger. What's happening?

A: Under CW7, you can debug only 68K code resources. Debugging PPC code resources will be available in CW8 (contact tech support to request a beta). If you are debugging a code resource under CW7, carefully follow the instructions in the Debugger manual on debugging code resources. Here is the basic procedure:

After creating the `.SYM` file for the code resource, change its name (to anything). Double-click on it, and then (since the name no longer corresponds to any executable), the debugger will ask you for the name of the executable to look at. Give it the name of the executable you've created that contains this resource. You can now set breakpoints. Next, leaving the `.SYM` window open, double-click on the application itself and control should be transferred to the Debugger. Let us know if this sequence of steps doesn't work. (In that case it might be necessary for us to look at a copy of the project in order to diagnose the problem.)

PROGRAMMER'S CHALLENGE

By Bob Boonstra, Westford, Massachusetts

Intersecting Rectangles

The Challenge this month is to write a routine that will accept a list of rectangles and calculate a result based on the intersections of those rectangles. Specifically, your code will return a list of non-overlapping rectangles that contain all points enclosed by an odd (or even) number of the input rectangles. The prototype for the code you should write is:

```
void RectangleIntersections(
  const Rect inputRects[],     /* list if input rectangles */
  const long numRectsIn,       /* number of inputRects */
  Rect outputRects[],          /* preallocated storage for output */
  long *numRectsOut,           /* number of outputRects returned */
  const Boolean oddParity      /* see text for explanation */
);
```

The parameter `oddParity` indicates whether you are to return rectangles containing points enclosed by an odd number of the `numRectsIn inputRects` rectangles (`oddParity==true`) or by an even (nonzero) number of rectangles (`oddParity==false`). Sufficient storage for the output will be preallocated for you and pointed to by `outputRects`.

As an example, if you were given these `inputRects`:

```
{0,10,20,30}, {5,15,20,30}
```

... and `oddParity` were true, you might return the following list of `outputRects`:

```
{0,10,5,15}, {0,15,5,30}, {5,10,15,20}
```

It would also be correct to return a result that combined the first of these rectangles with either of the other two. If `oddParity` were false, you would return the following list for the example input:

```
{5,15,20,30}
```

The `outputRects` must be non-empty and non-overlapping. In the example, it would be incorrect to return the following for the odd parity case:

```
{0,10,5,30} {0,10,20,15}
```

The `outputRects` you generate must also be maximal, in the sense that each edge of each of the `outputRects` should pass through a vertex of one of the `inputRects`. That is, for example, I don't want you to return a 1×1 rectangle representing each point enclosed in the desired number of `inputRects`. Before returning, set `*numRectsOut` to indicate the number of `outputRects` you generated.

If you need auxiliary storage, you may allocate any reasonable amount within your code using toolbox routines or malloc, but you must deallocate that storage before returning. (No memory leaks! – I'll be calling your code many times.)

This native PowerPC Challenge will be scored using the latest Metrowerks compiler, with the winner determined by execution time. If you have any questions, or would like some test data for your code, please send me e-mail at one of the Programmer's Challenge addresses, or directly to bob_boonstra@mactech.com. Test data will also be sent to the Programmer's Challenge mailing list, which you can join by sending a message to autoshare@mactech.com with the SUBJECT line "sub challenge YourName", substituting your real name for YourName.

Two Months Ago Winner

Eight of the 13 solutions submitted for the Find Again And Again Challenge worked correctly. Congratulations to **Gustav Larsson** (Mountain View, CA) for submitting an entry that was significantly

THE RULES

Here's how it works: Each month we present a new programming challenge. First, write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to MacTech magazine. We choose a winner based on code correctness, speed, size, and elegance (in that order of importance) as well as the submission date. In the event of multiple equally desirable solutions, we'll choose one winner (with honorable mention, but no prize, given to the runner up). The prize for each month's best solution is a $100 credit in the MacTech Mail Order Store and a limited-edition, "The Winner! MacTech Programmers Challenge" T-shirt (not available in stores anywhere).

Unless stated otherwise in the problem statement, the following rules apply: All solutions must be in ANSI compatible C. Use only pure C code. We disqualify entries with any assembly in them (except for challenges specifically stating otherwise). You may call any Macintosh Toolbox routine (e.g., it doesn't matter if you use NewPtr instead of malloc). We test entries with compiler options set to disable FPU use (for 680x0 code) and to enable all available speed optimizations. The compiler to be used and the target instruction set (680x0 or PowerPC) will be stated in the problem. **Limit your code to 60 characters per line;** this helps with e-mail gateways and page layout.

We publish the solution and winners for each month's Programmer's Challenge two months later. All submissions must be ***received by*** the 10th day of the month printed on the front cover of this issue.

You can get a head start on the challenge by reading the online version. We post it to the online services at the same time that we post source code. We make every effort to have it online no later than when the magazines are mailed, but we're unable to guarantee that it will be online by any given date.

Mark solutions "Attn: Programmer's Challenge Solution" and send it by e-mail to one of the Programmer's Challenge addresses in the "How to Communicate With Us" section on page 2 of this issue. Include the solution, all related files, and your contact info.

MacTech Magazine reserves the right to publish any solution entered in the Programmer's Challenge. Authors grant MacTech Magazine the exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

faster than the others. The problem was to write a text search engine optimized to operate repeatedly on the same block of text. A variety of optimization techniques were represented in the solutions, a couple of which are highlighted in the table of results below. Several people optimized for the case where the same word was repeatedly searched for. Some of my tests included this case, and those results are in the columns headed "repeat." The "random" columns shows results for tests that searched for random occurrences of random words. Each of the tests were run under conditions where only 64KB of auxiliary storage was available, and where much more memory was available. These conditions were weighted 20% and 80% respectively in calculating the total time, since the problem statement promised that ample memory would usually be provided. You can see that Gustav's solution performed reasonably well when memory was scarce, and very well when memory was plentiful.

Gustav's solution hashes as many words of the input text as possible in the initialization routine. He uses the Boyer-Moore-Horspool algorithm to find words in any text that was not parsed during initialization. Other features of the approach are described in the well-commented code.

Here are the times and code sizes for entries that passed by tests. Numbers in parentheses after a person's name indicate that person's cumulative point total for all previous Challenges, not including this one.

	64K Memory		>>64K Memory			code
Name	**repeat**	**random**	**repeat**	**random**	**time**	**size**
Gustav Larsson (67)	1814	3773	62	111	1255	3584
Tom Saxton	46	16400	197	459	3814	2000
Xan Gregg (81)	27	2907	1316	2835	3907	1664
Kevin Cutts (46)	1760	3234	1760	2809	4654	1600
Joseph Ku	8856	14570	121	509	5189	1584
David Cary	60	22665	499	1000	5745	2124
Eric Lengyel (40)	34	10221	29	4697	5831	1188
Ernst Munter (110)	2036	2053	2287	4603	6330	2976

Top Contestants of All time

Here are the Top Contestants for the Programmer's Challenges to date, including everyone who has accumulated more than 20 points. The numbers below include points awarded for this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	[Name deleted]	176	11.	Mallett, Jeff	44
2.	Munter, Ernst	110	12.	Kasparian, Raffi	42
3.	Gregg, Xan	88	13.	Vineyard, Jeremy	42
4.	Larsson, Gustav	87	14.	Lengyel, Eric	40
5.	Karsh, Bill	80	15.	Darrah, Dave	31
6.	Stenger, Allen	65	16.	Landry, Larry	29
7.	Riha, Stepan	51	17.	Elwertowski, Tom	24
8.	Cutts, Kevin	50	18.	Lee, Johnny	22
9.	Goebel, James	49	19.	Noll, Robert	22
10.	Nepsund, Ronald	47			

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place .....20 points	5th place . . . . . .2 points
2nd place.....10 points	finding bug . . . . . .2 points
3rd place .....7 points	suggesting Challenge .2 points
4th place.....4 points	

Here is Gustav's winning solution:

Find Again and Again

Constants & Types

```
#define ALPHABET_SIZE 256
#define ALLOC_SIZE(n) ((n+3) & ~4L) /* next multiple of 4 */
#define HASH_BUCKETS 1024           /* must be power of 2 */
#define HASH_MASK (HASH_BUCKETS - 1)
#define NO_NULL_CHAR 'A'
#define NULL 0

typedef unsigned char  uchar;
typedef unsigned short ushort;
typedef unsigned long  ulong;

typedef struct Word Word;
typedef struct Occurrence Occurrence;
typedef struct Private Private;
```

```
/*
A block of occurrence positions. We pack in as many occurrences as possible into
a single block, from 3 to 6 depending on textLength.

The first entry in the block is always used. The remaining entries are in use if they
are not zero. These facts are used several places to simplify the code.
*/
struct Occurrence {
  Occurrence *next;
  union {
    ushort pos2[6];     /* 2 bytes/occurrence */
    struct {
      ushort lo[4];
      uchar  hi[4];
    } pos3;             /* 3 bytes/occurrence */
    long pos4[3];       /* 4 bytes/occurrence */
  } p;
};

/*
There is one Word struct for each distinct word. The word's length is stored in the
top eight bits of the hash value. There's no need to store the characters in the word
since we can just look at the first occurrence (first entry in Word.first).
*/
struct Word {
  Word        *next;
  ulong       hash;
  Occurrence  *last;
  Occurrence  first;
};

/*
The structure of our private storage. The hashCodes[] array serves two purposes: it
distinguishes alphanumeric from non-alphanumeric characters, and it provides a
non-zero hash code for each alphanumeric character. The endParsedText field will
be -1 if there was enough private memory to parse all the text. Otherwise, it points to
the start of the unparsed text. nullChar is used by the BMH_Search() function when
we must search unparsed text for an occurrence.
*/
struct Private {
  ulong hashCodes [ ALPHABET_SIZE ];
  Word  *hashTable [ HASH_BUCKETS ];
  long  endParsedText;  /* start of parsed text */
  long  posBytes;       /* POS_x_BYTES, below */
  char  nullChar;       /* char not appearing in the text */
  long  heap;           /* start of private heap */
};
```

Macros

```
/*
These macros simplify access to the occurrence positions stored in an Occurrence
struct. Posbytes is a macro argument that is usually set to private->posBytes.
However, you can also use a constant for posbytes, which lets the compiler choose
the right case at compile time, producing smaller and faster code.
*/
#define POS_2_BYTES 1   /* word position fits in 2 bytes */
#define POS_3_BYTES 0   /* fits in 3 bytes; usual case */
#define POS_4_BYTES 2   /* fits in 4 bytes */

#define GET_POS(pos,occur,index,posbytes)                \
  {                                                      \
    if ( (posbytes) == POS_3_BYTES )                     \
      pos = ((long)(occur)->p.pos3.hi[index] << 16)      \
          + (occur)->p.pos3.lo[index];                   \
    else if ( (posbytes) == POS_2_BYTES )                \
      pos = (occur)->p.pos2[index];                      \
    else                                                 \
      pos = (occur)->p.pos4[index];                      \
  }

#define SET_POS(pos,occur,index,posbytes)                \
  {                                                      \
    if ( (posbytes) == POS_3_BYTES )                     \
    {                                                    \
      (occur)->p.pos3.hi[index] = (pos) >> 16;           \
      (occur)->p.pos3.lo[index] = (pos);                 \
    }                                                    \
    else if ( (posbytes) == POS_2_BYTES )                \
      (occur)->p.pos2[index] = pos;                      \
    else                                                 \
      (occur)->p.pos4[index] = pos;                      \
  }
```

InitFind

```
void InitFind (
  char *textToSearch,
  long textLength,
  void *privateStorage,
  long storageSize
)
{
  Private *private = privateStorage;

  private->endParsedText =
    InitFindBody(
        (uchar *)textToSearch,
        textLength,
        privateStorage,
        (uchar *)privateStorage + storageSize
    );

  if ( private->endParsedText != -1 )
    private->nullChar =
      PickNullChar(
          private,
          (uchar *)textToSearch + private->endParsedText,
          (uchar *)textToSearch + textLength );
  else
    private->nullChar = NO_NULL_CHAR;
}
```

InitFindBody

```
/*
This function does most of the work for InitFind(). The arguments have been recast
into a more useful form; uchar and ulong are used a lot so that we don't have to
worry about the sign, especially when indexing private->hashCodes[].

The return value is the character position when the unparsed text begins (if we run
out of private storage), or -1 if all the text was parsed.
*/
static long InitFindBody (
  uchar   *textToSearch,
  long    textLength,
  Private *private,
  uchar   *endPrivateStorage
)
{
  uchar       *alloc, *textPos, *textEnd, *wordStart;
  long        wordLength;
  ulong       hash, code;
  Word        *word;
  Occurrence  *occur;

/*
Init table of hash codes. The remaining entries are guaranteed to be initialized to
zero. The hash codes were chosen so that any two codes differ by at least five bits.
*/
  {
    ulong *table = private->hashCodes;  /* reduces typing */

    table['0'] = 0xFFC0;  table['5'] = 0xF492;
    table['1'] = 0xFE07;  table['6'] = 0xF31E;
    table['2'] = 0xF98B;  table['7'] = 0xF2D9;
    table['3'] = 0xF84C;  table['8'] = 0xCF96;
    table['4'] = 0xF555;  table['9'] = 0xCE51;

    table['A'] = 0xC9DD;  table['N'] = 0xA245;
    table['B'] = 0xC81A;  table['O'] = 0x9F0A;
    table['C'] = 0xC503;  table['P'] = 0x9ECD;
    table['D'] = 0xC4C4;  table['Q'] = 0x9941;
    table['E'] = 0xC348;  table['R'] = 0x9886;
    table['F'] = 0xC28F;  table['S'] = 0x959F;
    table['G'] = 0xAF5C;  table['T'] = 0x9458;
    table['H'] = 0xAE9B;  table['U'] = 0x93D4;
    table['I'] = 0xA917;  table['V'] = 0x9213;
    table['J'] = 0xA8D0;  table['W'] = 0x6DD3;
    table['K'] = 0xA5C9;  table['X'] = 0x6C14;
    table['L'] = 0xA40E;  table['Y'] = 0x6B98;
    table['M'] = 0xA382;  table['Z'] = 0x6A5F;

    table['a'] = 0x6746;  table['n'] = 0x3C88;
```

```
    table['b'] = 0x6681;  table['o'] = 0x3B04;
    table['c'] = 0x610D;  table['p'] = 0x3AC3;
    table['d'] = 0x60CA;  table['q'] = 0x37DA;
    table['e'] = 0x5D85;  table['r'] = 0x361D;
    table['f'] = 0x5C42;  table['s'] = 0x3191;
    table['g'] = 0x5BCE;  table['t'] = 0x3056;
    table['h'] = 0x5A09;  table['u'] = 0x0D19;
    table['i'] = 0x5710;  table['v'] = 0x0CDE;
    table['j'] = 0x56D7;  table['w'] = 0x0B52;
    table['k'] = 0x515B;  table['x'] = 0x0A95;
    table['l'] = 0x509C;  table['y'] = 0x078C;
    table['m'] = 0x3D4F;  table['z'] = 0x064B;
}

/* Determine the number of bytes needed to store each occurrence position. */
if ( textLength <= 0x10000L )
  private->posBytes = POS_2_BYTES;
else if ( textLength <= 0x1000000L )
  private->posBytes = POS_3_BYTES;
else
  private->posBytes = POS_4_BYTES;

/* Set up variables to handle allocation of private storage. */
alloc = (uchar *)&private->heap;

/* Parse the text */
textPos = textToSearch;
textEnd = textPos + textLength;

while ( textPos != textEnd )
{
/* Search for start of word */
  while ( private->hashCodes[*textPos] == 0 )
  {
    textPos++;
    if ( textPos == textEnd )
      return -1;  /* parse all text */
  }
  wordStart = textPos;

/* Search for end of word; generate hash value too */
  hash = 0;
  while ( textPos != textEnd &&
          (code = private->hashCodes[ *textPos ]) != 0 )
  {
    hash = (hash << 1) ^ code;
    textPos++;
  }
  wordLength = textPos - wordStart;
  hash = (hash & 0xFFFFFF) | (wordLength << 24);

/*
Record the occurrence. First we see if a Word struct exists for this word and whether we need to allocate a new Occurrence struct.
*/
  word = LookupWord(
              private,
              (char *)textToSearch,
              (char *)wordStart,
              wordLength,
              hash );
  if ( word )
  {
    long allocateNewBlock, blockSize, i, pos;

/*
This word has occurred before, so it already has a Word struct. See if there's room in the last Occurrence block for another entry. Remember that entry #0 in the Occurrence block is always in use, so we can start checking at entry #1 for a non-zero entry.
*/
    occur = word->last;
    allocateNewBlock = TRUE;
    switch ( private->posBytes )
```

```
    {
      case POS_2_BYTES:  blockSize = 6; break;
      case POS_3_BYTES:  blockSize = 4; break;
      case POS_4_BYTES:  blockSize = 3; break;
    }

    for ( i = 1; i < blockSize; i++ )
    {
      GET_POS( pos, occur, i, private->posBytes )
      if ( pos == 0 )
      {
        SET_POS( wordStart - textToSearch, occur, i,
                 private->posBytes )
        allocateNewBlock = FALSE;
        break;
      }
    }

    if ( allocateNewBlock )
    {
  /* Block is full.  Allocate new Occurrence block */
      occur = (Occurrence *) alloc;
      alloc += ALLOC_SIZE( sizeof(Occurrence) );
      if ( alloc >= endPrivateStorage )
        return wordStart-textToSearch; /* out of memory */

  /* Init the new struct and link it to the end of the occurence list. */
      SET_POS( wordStart - textToSearch, occur, 0,
               private->posBytes )
      word->last->next = occur;
      word->last = occur;
    }
  }
  else
  {
    long i;

/* This is a new word.  Allocate a new Word struct, which contains an Occurrence
   struct too. */
    word = (Word *) alloc;
    alloc += ALLOC_SIZE( sizeof(Word) );
    if ( alloc >= endPrivateStorage )
      return wordStart-textToSearch ;  /* out of memory */

  /*  Link it to the start of the Word list, coming off the hash table. */
    word->next = private->hashTable[ hash & HASH_MASK ];
    private->hashTable[ hash & HASH_MASK ] = word;

  /* Init the Word struct */
    word->hash = hash;
    word->last = &word->first;

  /* Init the Occurrence struct */
    SET_POS( wordStart - textToSearch, &word->first, 0,
             private->posBytes )
  }
}

/* Finished parsing text */
return -1;
}
```

FindWordOccurrence

```
long FindWordOccurrence (
  char *wordToFind,
  long wordLength,
  long occurrenceToFind,
  char *textToSearch,
  long textLength,
  void *privateStorage,
  long storageSize
)
{
  Private *private = privateStorage;
  Word  *word;
  ulong hash;

/* Make occurenceToFind zero-based */
  occurrenceToFind--;

/* Generate hash value for word to find */
  hash = 0;
  {
    long remain = wordLength;
    uchar *p = (uchar *) wordToFind;
    while ( remain > 0 )
    {
      hash = (hash << 1) ^ private->hashCodes[*p++];
      remain--;
    }
    hash = (hash & 0xFFFFFF) | (wordLength << 24);
  }

/* Look for word/occurrence in hash table */
  word = LookupWord( private, textToSearch, wordToFind,
                     wordLength, hash );
  if ( word )
  {
    Occurrence *occur = &word->first;
    long blockSize, pos, i;

  /* Word exists in hash table, so go down the occurrence list. */
    switch ( private->posBytes )
    {
      case POS_2_BYTES: blockSize = 6;  break;
      case POS_3_BYTES: blockSize = 4;  break;
      case POS_4_BYTES: blockSize = 3;  break;
    }

    while ( occur && occurrenceToFind >= blockSize )
    {
      occurrenceToFind -= blockSize;
      occur = occur->next;
    }

    if ( occur )
    {
      GET_POS( pos, occur, occurrenceToFind,
               private->posBytes )
      if ( occurrenceToFind == 0 || pos != 0 )
        return pos;
      occurrenceToFind -= blockSize;
    }

    occur = word->last;
    for ( i = 0; i < blockSize; i++ )
    {
      GET_POS( pos, occur, i, private->posBytes )
      if ( pos == 0 )
        occurrenceToFind++;
    }
  }

/* Not in parsed text, so check the unparsed text */
  if ( private->endParsedText != -1 )
  {
    char *p;
    if ( wordLength > 3 )
      p = BMH_Search(
              private->hashCodes,
              wordToFind,
              wordLength,
              occurrenceToFind,
              textToSearch + private->endParsedText,
              textToSearch + textLength,
              private->nullChar );
    else
      p = SimpleSearch(
              private->hashCodes,
              wordToFind,
              wordLength,
              occurrenceToFind,
              textToSearch + private->endParsedText,
              textToSearch + textLength );
    if (p)
      return (p - textToSearch);
  }

/* Not found */
  return -1;
}
```

LookupWord

```
/* Look up a word in the hash table */
static Word *LookupWord (
  Private *private,
  char    *textToSearch,
  char    *wordText,
  long    wordLength,
  ulong   hash
)
{
  Word *word = private->hashTable[ hash & HASH_MASK ];
  while ( word )
  {
    if ( word->hash == hash )
    {
      char *w1, *w2;
      long pos, remain = wordLength;

    /*
      The hash values match, so compare characters to make sure it's the right word.
      We already know the word length is correct since the length is contained
      in the upper eight bits of the hash value.
    */
      GET_POS( pos, &word->first, 0, private->posBytes )
      w1 = textToSearch + pos;
      w2 = wordText;
      while ( remain-- > 0 && *w1++ == *w2++ )
        ;
      if ( remain == -1 )
        return word;
    }
    word = word->next;
  }
  return NULL;
}
```

PickNullChar

```
/*
  Find a character that doesn't appear anywhere in the unparsed text.  BMH_Search() is
  faster if such a character can be found.
*/
static char PickNullChar (
  Private *private,
  uchar   *textStart,
  uchar   *textEnd
)
{
  long i;
  uchar *p, occurs[ ALPHABET_SIZE ];

  for ( i = 0; i < ALPHABET_SIZE; i++ )
    occurs[i] = FALSE;

  for ( p = textStart; p < textEnd; p++ )
    occurs[*p] = TRUE;

  for ( i = 0; i < ALPHABET_SIZE; i++ )
    if ( occurs[i] == FALSE && private->hashCodes[i] == 0 )
      return i;

  return NO_NULL_CHAR;
}
```

BMH_Search

```
/*
  Search the unparsed text using the Boyer-Moore-Horspool algorithm.  Ideally a null
  character is supplied (one that appears in neither the search string nor the text being
  searched).  This allows the inner loop to be faster.
*/
static char *BMH_Search (
  ulong *hashCodes,       /* private->hashCodes   */
  char  *wordToFind,
  long  wordLength,
  long  occurrenceToFind, /* 0 is first occurrence */
  char  *textStart,       /* start of unparsed text */
  char  *textEnd,         /* end of unparsed text  */
  char  nullChar          /* private->nullChar    */
)
{
  long  i;
  char  *text, *wordEnd;
  char  word[256];
  long  offset[ ALPHABET_SIZE ];

  /*
    Copy the search string to a private buffer, where
    the first character is the null character.
  */
  word[0] = nullChar;
  for ( i = 0; i < wordLength; i++ )
    word[i+1] = wordToFind[i];

  /* Set up the offset[] lookup table */
```

```
  for ( i = 0; i < ALPHABET_SIZE; i++ )
    offset[i] = wordLength;

  for ( i = 1; i < wordLength; i++ )
    offset[ word[i] ] = wordLength - i;

 /* Let the search begin... */
  wordEnd = word + wordLength;
  text = textStart + wordLength - 1;

  if ( nullChar == NO_NULL_CHAR )
  {
  /* No null character, so use a slower inner loop */
    while ( text < textEnd )
    {
      long i;
      char *p, *q;
      for ( i = wordLength, p = wordEnd, q = text;
            i > 0 && *p == *q;
            i--, p--, q-- )
        ;
/*If i == 0, we have found the search string.  Now we make sure that it is delimited.*/
      if ( i == 0 && hashCodes[*q] == 0 &&
           (text+1 == textEnd || hashCodes[text[1]] == 0) )
      {
        if ( occurrenceToFind == 0 )
          return q+1;
        occurrenceToFind--;
      }

      text += offset[*text];
    }
  }
  else
  {
  /* There is a null character (usual case),
     so we can use a faster and simpler inner loop. */
    while ( text < textEnd )
    {
      char *p, *q;
      for ( p = wordEnd, q = text; *p == *q; p--, q-- )
        ;
      if ( p == word && hashCodes[*q] == 0 &&
           (text+1 == textEnd || hashCodes[text[1]] == 0) )
      {
        if ( occurrenceToFind == 0 )
          return q+1;
        occurrenceToFind--;
      }
      text += offset[*text];
    }
  }
  return NULL;
}
```

SimpleSearch

```
/*
 Search the unparsed text using a simple search algorithm.  Note that wordLength
 must be 1, 2, or 3.  This algorithm runs faster than BMH_Search() for small search
 strings.
*/
static char *SimpleSearch(
  ulong *hashCodes,        /* private->hashCodes    */
  char  *wordToFind,
  long  wordLength,        /* 1..3                  */
  long  occurrenceToFind,  /* 0 is 1st occurrence   */
  char  *textStart,        /* start of unparsed text */
  char  *textEnd           /* end of all text       */
)
{
  char *text, first;

  first = wordToFind[0];
  text = textStart;

  if ( wordLength == 1 )
  {
    while ( text < textEnd )
    {
      while ( text < textEnd && *text != first )
        text++;
```

```
    if ( hashCodes[*(text-1)] == 0 &&
         hashCodes[text[wordLength]] == 0 )
    {
      if ( occurrenceToFind == 0 )
        return text;
      occurrenceToFind--;
    }
  text++;
  }
}
else if ( wordLength == 2 )
{
  while ( text < textEnd )
  {
    while ( text < textEnd && *text != first )
      text++;
    if ( text[1] == wordToFind[1] &&
         hashCodes[*(text-1)] == 0 &&
         hashCodes[text[wordLength]] == 0 )
    {
      if ( occurrenceToFind == 0 )
        return text;
      occurrenceToFind--;
    }
  text++;
  }
}
else /* wordLength == 3 */
{
  while ( text < textEnd )
  {
    while ( text < textEnd && *text != first )
      text++;
    if ( text[1] == wordToFind[1] &&
         text[2] == wordToFind[2] &&
         hashCodes[*(text-1)] == 0 &&
         hashCodes[text[wordLength]] == 0 )
    {
      if ( occurrenceToFind == 0 )
        return text;
      occurrenceToFind--;
    }
  text++;
  }
}
return NULL;
}
```

By Jeremy Roschelle

Attaching a Scripts Menu

An introduction to using the OSA in PowerPlant

A fully AppleEvent-savvy application is scriptable, recordable, and attachable. In a *scriptable* application, any user can automate tasks, interconnect applications, and extend the capabilities of your application. A *recordable* application generates a script by observing the user's actions. Yet these capabilities prove worthless if users have no easy way to *execute* their scripts. Unfortunately, Apple did not provide any standard human interface for attaching scripts to an application. And although the PowerPlant framework provides excellent support for scripting and recording, it provides no recipes for customizing your application to launch scripts. This article addresses these issues with a simple customizable **Script** menu which allows users to execute scripts (see Figure 1).

At first, implementing a script menu looks complex: it requires interacting with PowerPlant, the Menu Manager, the File Manager, the Open Scripting Architecture (OSA), and the scriptable Finder. On the bright side, PowerPlant and the OSA provide excellent modular, easy-to-use interfaces [see Jeremy's article, "Powering Up AppleEvents in PowerPlant", *MacTech Magazine,* 11:6 (1994) 33–46 – *man*]. In this article, I'll present an implementation of an extension to the PowerPlant framework that that can compile and execute scripts from a standard pull-down menu. This script menu provides a relatively complete implementation of attachable scripts: it loads scripts at launch time from a "script menu items" folder, automatically supplies Balloon Help for each menu item, and can open a script for editing in the Script Editor. This article will also show you how easy it is to use OSA to compile and execute scripts.

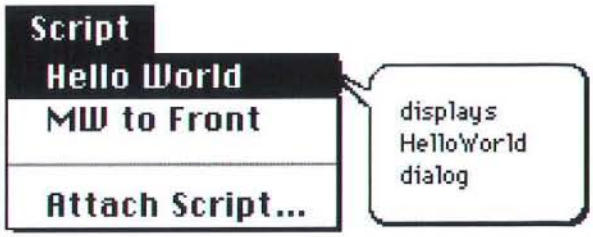


Figure 1. The Script menu

The implementation also strives to use the capabilities of PowerPlant, C++, and AppleEvents to achieve modularity and encapsulation. For example, we use the `LAttachment` mechanism to encapsulate all the code for handling the script menu into a re-usable class. Likewise, we introduce a C++ iterator class for scanning through items in a folder. Finally, we use AppleEvents to connect our script menu to external applications that provide script editing services, thus avoiding the need to embed a script editor ourselves.

Jeremy Roschelle – Jeremy Roschelle works for the University of Massachusetts, at the, umm, "San Francisco campus." From his spacious, bedroom-like, corner office, he is developing software to help students learn "the mathematics of change" with interactive graphs and animated characters. Crossing "Doom" with Calculus provides a positive outlet for responding to traumatic memories of past mathematics teachers. And because his Ph.D. is in education, there's hardly anything else he's qualified to do. Contact jeremy at jeremy@dewey.soe.berkeley.edu.

OSA Basics

From the user's point of view, a script is a small program written in AppleScript or another OSA dialect. From the programmer's point of view, a script is a data type containing code that the OSA can execute. The process of *compiling* a script reconciles these views: compiling converts AppleScript statements into an executable data type.

As a programmer, you manipulate a script as a Handle to some script data. The easiest way to get such a handle is to get the `'scpt'` resource out of a ScriptEditor document. There is one `'scpt'` resource in each ScriptEditor document, storing the script compiled by the user.

To run a script, you first *load* the script data into OSA. This results in an OSAID, a token that refers to the loaded script. The process of loading is relatively slow (a second or two on my Quadra 660AV). Once a script is loaded, running it is fast (small scripts seem as fast as hard-coded commands in my application). To *execute* the script, you pass the OSAID to the `OSAExecute` function. When you are through with a script (or any value returned by OSA), you dispose of its OSAID to free up the associated memory.

To hide the ugly details, I wrapped my code in a utility class with static methods:

UScripting::Initialize

```
void UScripting::Initialize()
{
  // sComponent is a static class member of type ComponentInstance
  if (sComponent == nil)
    SetComponent(
      ::OpenDefaultComponent(kOSAComponentType, 'scpt'));
}
```

UScripting::LoadScript

```
OSErr UScripting::LoadScript(
    Handle inScript,
    OSAID  &outScriptID)
{
  Initialize();

  AEDesc scriptDesc;
  scriptDesc.descriptorType = typeOSAGenericStorage;
  scriptDesc.dataHandle = inScript;
  return ::OSALoad(sComponent,
                   &inScriptDesc,
                   kOSAModeNull,
                   &outScriptID);
}
```

UScripting:: ExecuteScript

```
OSErr UScripting::ExecuteScript(
    OSAID inScriptID)
{
  Initialize();

  OSAID  resultID;
  OSErr  err;

  err = ::OSAExecute(sComponent,
                     inScriptID,
                     kOSANullScript,
                     kOSAModeNull, &resultID);
  if (err)
    return err;
  else ::OSADispose(sComponent, resultID);
  return noErr;
}
```


Design Overview

The main challenge in designing a script menu is maintaining a correspondence between items in a Menu Manager menu and script data that we can execute. This script data (an FSSpec for a script file and an OSAID for an executable script) will be encapsulated by a class called `SCScriptMenuItem`. Because scripts will be added to the menu dynamically, we cannot specify the menu items ahead of time in our resource file and cannot use PowerPlant's `'Mcmd'` scheme for binding each menu item to a command number. Instead, the implementation builds a list of `SCScriptMenuItems`, where the index of the item in the list matches the index of the item in the menu.

Our application must use this correspondence to respond when the user selects an item from the **Script** menu. We could do this by overriding `LApplication` methods that handle menu commands. But PowerPlant's `LAttachment` class provides a better solution. It allows the code to be completely encapsulated in a class, `SCScriptMenuHandler`. This class can be attached to *any* PowerPlant application with one `AddAttachment` call. (Such modularity and portability can be dangerous – your employer may come to expect it regularly!)

Your application will normally create one `SCScriptMenuHandler` at launch time. When created, this object will iterate through the designated folder and create one

MacOS menu item and a corresponding `SCScriptMenuItem` for each script in the folder. When a user selects a script from the menu for execution or editing, the `SCScriptMenuHandler` calls the appropriate method of the corresponding `SCScriptMenuItem`.

The article covers the implementation starting from the basic structure of `SCScriptMenuHandler` and `SCScriptMenuItem`. Next, the article describes how to create Balloon Help for each script automatically. Finally, the article reviews the utility routines for interacting with the File Manager.

Creating the Script Menu

Like every menu, the **Script** menu requires a `'MENU'` resource, a `'hmnu'` resource for Balloon Help, and a reference to the correct ID in your `'MBAR'` resource. The `'MENU'` and `'hmnu'` resources contain the fixed portion of the script menus: the menu title and a final menu item that allows the user to add a script to the menu while your application is running. (This additional feature is supported in the sample code, but not discussed in this article.) At run-time, we add additional menu and help items for each script.

In your application, you create a handler for this script menu, normally within the constructor for your application class. When creating the handler, you provide the resource id for the script menu, and the vRefNum and dirID for the folder from which you wish to load scripts.

YourApp constructor

```
YourApp::YourApp()
{
  // get folder id and volume number for the Scripts Folder, relative to launch spec
  FSSpec appSpec;
  long   folderID;

  UFinder::GetAppSpec(appSpec);
  folderID = UFinder::GetFolderID(
    appSpec, "\pScript Menu Items");

  // attach a new handler for the scripts menu
  AddAttachment(
    new SCScriptsMenuHandler( kScriptsMenuID,
                              appSpec.vRefNum,
                              folderID));
}
```

When `SCScriptsMenuHandler` is constructed, it iterates through a folder, appending a script menu item for each script file it finds. To hide the ugly details of iterating through a folder, the implementation uses an iteration class, `StFolderIterator`.

SCScriptsMenuHandler constructor

```
SCScriptsMenuHandler::SCScriptsMenuHandler(
        ResIDT inMenuID,
        short  inVRefNum,
        long   inParID,
        Int16  inMax)
  : LAttachment(msg_AnyMessage, true), mMenuID(inMenuID)
{
  // appends menu items for each script in the designated folder
  if (inVRefNum != 0) {
    // set up iteration structs
    Int16       count = 0;
    Str255      scriptFileName;
    HFileParam fInfo;
    fInfo.ioNamePtr = scriptFileName;

    // iterate through each item in the folder, inserting scripts
    StFolderIterator  iter(inVRefNum, inParID);
    while ((++count <= inMax) && iter.Next(fInfo)) {
      if (fInfo.ioFlFndrInfo.fdType == kOSAFileType) {
        FSSpec spec;
        FSMakeFSSpec(
          inVRefNum, inParID, scriptFileName, &spec);
        AppendScript(spec);
      }
    }
  }
}
```

To append each script, we first grab the menu. Then we insert an item into the menu, using the file name as the menu item name. To handle each menu item, we build a `SCScriptMenuItem` and insert it in the `mScripts` list, such that index numbers of the MacOS menu item and the `SCScriptMenuItem` correspond. Finally, we construct Balloon Help (as described later).

SCScriptsMenuHandler::AppendScript

```
void SCScriptsMenuHandler::AppendScript(
    FSSpec &inScriptFile)
{
  MenuHandle menu = ::GetMenu(mMenuID);
  if (! menu) return;

  SCScriptsMenuItem *item =
    new SCScriptsMenuItem(inScriptFile);

  // insert into the menu
```

```
    ::InsMenuItem(
        menu, inScriptFile.name, mScripts.GetCount());

    // insert the corresponding class instance into the list
    mScripts.InsertItemsAt(1, arrayIndex_Last, &item);

    // insert balloon help into resource
    AttachBalloonHelp(inScriptFile, mScripts.GetCount());
}
```

Running a Script

As described earlier, running a script in the OSA requires two simple steps. First you *load* the script, resulting in token called an OSAID that represents the executable. Then you pass the token to the OSA *execute* function.

Running scripts from a menu is only slightly more complicated. The `AppendScript` procedure created a `SCScriptMenuItem` for each menu item, storing the FSSpec of a script file. To compile a script, we need to extract the `'scpt'` resource from this file and pass it to `OSALoad` to get an OSAID. Because loaded scripts execute much faster, we load the script and store the OSAID to service future requests to run the same script.

SCScriptsMenuHandler:: RunScript

```
OSErr SCScriptsMenuItem::RunScript()
{
  OSErr  err = noErr;
  // load the script if its not available yet
  if (mScriptID == kOSANullScript) {
    Handle   script = nil, text = nil;
    short    fRefNum = -1;

    Try_ {
      // open resource fork
      fRefNum = ::FSpOpenResFile(&mFileSpec, fsRdPerm);
      ThrowIfResError_();
      // get the first script resource in the file
      script = ::Get1IndResource('scpt', 1);
      FailNIL_(script);
      // Load it
      UScripting::LoadScript(script, mScriptID);
    }
    Catch_(catchErr) {
      err = catchErr;
      SysBeep(0);
    }
    EndCatch_
    if (fRefNum != -1) ::CloseResFile(fRefNum);
  }
  if (err == noErr) new URun1Script(mScriptID);
  return err;
}
```

Testing reveals one additional complication. If the script brings a different application to the front while you are still handling a menu selection, a menubar drawing glitch occurs. To solve this problem, we create a `LPeriodical` task that runs immediately after the menu event completes (and the MacOS has removed the menu hiliting). `URun1Script` simply executes a loaded script with a given OSAID and then deletes itself.

URun1Script constructor

```
URun1Script::URun1Script(OSAID inScriptID)
  : mScriptID(inScriptID)
{
  StartRepeating();
}
```


URun1Script::SpendTime

```
void URun1Script::SpendTime(
    const EventRecord &inMacEvent)
{
  UScripting::ExecuteScript(mScriptID);
  delete this;
}
```

Handling The Menu Selection

Handling menu selection in an `LAttachment` is a matter of overriding `ExecuteSelf`. When the user selects the menu item, PowerPlant will generate a negative command number (because the menu has no `'Mcmd'` resource). The menu id will be in the HiWord, and the item number in the LoWord.

Our handler must respond both to this command and to a command status message that enables the menu item. Since scripts are always available, we enable all menu items in the script menu. To respond to the command, we find the corresponding `SCScriptMenuItem`. Normally we run the script. However, if the command key is down we open it for editing. The methods for running a script were described above; the next section explains how to open a script.

SCScriptsMenuHandler::ExecuteSelf

```
void SCScriptsMenuHandler::ExecuteSelf(
    MessageT inMessage,
    void     *ioParam)
{
```

```
    mExecuteHost = true;
    // update status
    if (inMessage == msg_CommandStatus) {
      SCommandStatus  *status = (SCommandStatus *)ioParam;
      if (HiWord(*status->command) == mMenuID) {
        *status->enabled = true;
        *status->usesMark = false;
        mExecuteHost = false; // we handled it
      }
    }
    // handle menu comand
    else if (HiWord(inMessage) == mMenuID) {
      Int16  index = LoWord(inMessage);
      SCScriptsMenuItem *item;
      if (mScripts.FetchItemAt(index, &item)) {
        if (cmdKey & UEventUtils::GetModifiers())
          item->OpenScript(); // open on command key
        else item->RunScript();
        mExecuteHost = false; // we handled it
        }
      }
    }
}
```

Editing a Script, the AppleEvent Way

Providing support for users to edit scripts is not hard. OSA provides calls that get the text and style record for a script, which you can display in an `LTextEdit` pane. When the user finishes her changes, you can use OSA calls to compile the script, and then execute it. But there is an easier way: the ScriptEditor already provides full script editing capabilities. By sending an AppleEvent, we can open a file in ScriptEditor and let it handle editing.

Since we already have an FSSpec for each script in our menu, this is easy. Our `SCScriptMenuItem` method for opening a script calls a utility method to send the Finder an "open" event with the FSSpec. Before doing so, we dispose of the token that represents the loaded script. By doing this, we will force our `RunScript` method to re-load the script from the file. Thus, when the user edits and then saves the script, her next attempt to run it will load and execute the modified version.

SCScriptsMenuItem::OpenScript

```
OSErr SCScriptsMenuItem::OpenScript()
{
  if (mScriptID != kOSANullScript) {
    // first unload script from OSA
    UScripting::DisposeScript(mScriptID))
    mScriptID = kOSANullScript;
  }
  return UFinder::SendFinderAEOpen(mFileSpec);
}
```

We could send an "open" event to ScriptEditor, but instead we send it to the scriptable Finder. The Finder will open the correct script editing application based on the creator of the file.

Sending an AppleEvent is not hard. The first step is to create a descriptor for the target of the event, in this case the Finder. The easiest type of process descriptor just uses the application signature. The second step is to create an AppleEvent with this process descriptor. The third step is to add any parameters to the event. In this case there is just one, the FSSpec. Finally we send the event and dispose of the reply.

The implementation uses exceptions to handle an error at any stage of the process, but it catches all errors, disposes of the memory in AEDescs and returns the error code.

UFinder::SendFinderAEOpen

```
OSErr UFinder::SendFinderAEOpen(
    FSSpec &inFile)
{
  OSErr       err = noErr;
  AEDesc      processDesc;
  AppleEvent ae, aeReply;
  ae.descriptorType =
    aeReply.descriptorType =
    processDesc.descriptorType = typeNull;
  ae.dataHandle =
    aeReply.dataHandle =
    processDesc.dataHandle =
    nil;

  Try_ {
    DescType finderType = 'MACS';
    err = ::AECreateDesc(
          typeApplSignature,
          &finderType,
          sizeof(DescType),
          &processDesc);
    FailOSErr_(err);

    err = ::AECreateAppleEvent(
          kCoreEventClass,
          kAEOpen,
          &processDesc,
          kAutoGenerateReturnID,
          kAnyTransactionID,
          &ae);
    FailOSErr_(err);
```

```
    err = ::AEPutParamPtr(
          &ae,
          keyDirectObject,
          typeFSS,
          &inFile,
          sizeof(inFile));
    FailOSErr_(err);
    err = ::AESend(
          &ae,
          &aeReply,
          kAENoReply | kAENeverInteract,
          kAENormalPriority,
          kAEDefaultTimeout,
          nil,
          nil);
    FailOSErr_(err);
  }
  Catch_(catchErr) { err = catchErr; } EndCatch_

  if (processDesc.descriptorType != typeNull)
    ::AEDisposeDesc(&processDesc);
  if (ae.descriptorType != typeNull)
    ::AEDisposeDesc(&ae);
  if (aeReply.descriptorType != typeNull)
    ::AEDisposeDesc(&aeReply);
  return err;
}
```

Writing Balloons Without Typing

As a final touch, it's nice to provide Balloon Help for all menu items. But scripts are loaded at run time, so there's no way to know in advance what scripts will be present. Yet there is a way to automatically create sensible help text for each script at run time. Here's how.

When a user creates a script in ScriptEditor, the user can write an English description of the script in the area just below the window title. This description ends up in a `'TEXT'` resource in the script file. The script menu can grab this text from the file, truncate it to 255 characters, and install it as Balloon Help for the menu item. Thus, the Script Editor description field becomes the Balloon Help automatically.

Here is the top-level routine that is called when the `SCScriptsMenuHandler` is constructed.

SCScriptsMenuHandler::AttachBalloonHelp

```
void SCScriptsMenuHandler::AttachBalloonHelp(
    FSSpec &inScriptFile,
    Int16  inIndex)
{
  Str255 text;
  {
    // get the text
    Int16  fRefNum =
      ::FSpOpenResFile(&inScriptFile, fsRdPerm);
    if (ResError()) return;

    // the first text resource has the description of the script
    Handle outText = ::Get1IndResource('TEXT', 1);
    if (outText)
      UFinder::Handle2PStr(outText, text);
    else *text = 0;

    ::CloseResFile(fRefNum);
  }

  {
    // add the help
    char   buffer[500];
    MakeBalloonData(text, buffer);
    InsertBalloonData(inIndex, buffer);
  }
}
```


Once we have extracted the description text, the process of installing it is divided into 2 steps. First we construct a buffer containing a single entry for the `'hmnu'` resource. Each entry begins with a size word for the size of the entry, and then a flag word indicating the type of the entry. We only deal with two kinds of entries, a "skip" entry for empty balloons, and a direct string entry. A direct string entry has 4 packed Pascal strings. The routine below writes an entry in this format, implementing the writes as if writing to a stream.

SCScriptsMenuHandler::MakeBalloonData

```
void SCScriptsMenuHandler::MakeBalloonData(
    Str255 inHelp,
    char   *ioBuffer)
{
  Int16  mark, data;
  Int32  zeros = 0;

  // leave room to write number of bytes to end
  mark = 2;

  if (*inHelp == 0) {
    // no data, so skip this item
    data = 0x0100;
    ::BlockMoveData(&data, ioBuffer[mark], sizeof(Int16));
    mark += sizeof(Int16);
  }
  else {
    data = 0x0001; // direct string type
    ::BlockMoveData(&data, &ioBuffer[mark], sizeof(Int16));
```

```
    mark += sizeof(Int16);

    // write out the string
    ::BlockMoveData(inHelp, &ioBuffer[mark], 1 + *inHelp);
    mark += 1 + *inHelp;

    // write out three zeros for the other strings
    ::BlockMoveData(&zeros, &ioBuffer[mark], 3);
    mark += 3;
  }

  // align buffer to an even word boundary
  if (mark & 0x0001) ++mark;

  // add size to first word of buffer
  ::BlockMoveData(&mark, ioBuffer, sizeof(mark));
}
```

Balloon data for a menu is packed into a single Handle. In order to insert an entry for a new menu item, we need to increment the count word, and then insert the entry in the right place. To find the right place we have to read the size of each preceding entry, and skip over that many bytes to arrive at the next entry. Once we find the right place, remaining entries are moved out of the way, and the new entry is copied into place.

SCScriptsMenuHandler:: InsertBalloonData

```
void SCScriptsMenuHandler::InsertBalloonData(
    Int16  inIndex,
    char   *inBuffer)
{
  Handle hmnu = ::GetIResource('hmnu', mMenuID);
  if (! hmnu) return;

  Int16  len = *(short *)inBuffer;

  // make some room in the handle
  ::SetHandleSize(hmnu, ::GetHandleSize(hmnu) + len);
  if (::MemError()) return;

  // lock it down so we can safely dereference it
  StHandleLocker  lock(hmnu);
  char   *help = *hmnu;

  // increment number of items
  ++*(short *)(help + 0x0A); // @ help + 0x0A

  // skip over existing items
  {
    // skip default and title resource, don't skip self
    Int16  itemsToSkip = inIndex + 2 - 1;
    help += 0x0C; // location of first msg record
    do {
      help += *(Int16 *)help;  // add the number of bytes to skip
    } while (--itemsToSkip);
  }

  // shift data out of the way
  {
    char *dest, *end;
    dest = help + len;
    end = ((char *)*hmnu + ::GetHandleSize(hmnu));
    ::BlockMoveData(help, dest, end - dest);
  }

  // copy help data in
  ::BlockMoveData(inBuffer, help, len);
}
```

Note that the implementation does *not* call ChangedResource, even though it did change the resource. This is because the resource is in the application, and calling ChangedResource would cause the application to store the Balloon data when it quit. We don't want this data stored; it is re-computed every time the application is launched. We also don't call ReleaseResource, so the changed resource will stay in memory for the duration of the session.

Finder Utilities

The implementation made use of a few Finder utilities: (a) for finding the FSSpec of the running application; (b) for finding a folder id, given a parent folder and a folder name; (c) for iterating through all the items in a folder. These are fairly common steps in many applications, but the techniques are not easy to find in standard Macintosh references. For the sake of completeness, the routines are presented below:

To find the FSSpec of the running application, you call the process manager, requesting information about the current process.

UFinder::GetAppSpec

```
UFinder::GetAppSpec(
    FSSpec &inSpec)
{
  ProcessSerialNumber psn;
  ProcessInfoRec      info;
  info.processAppSpec = &inSpec;
  info.processInfoLength = sizeof(info);
  info.processName = nil;
  ::GetCurrentProcess(&psn);
  ::GetProcessInformation(&psn, &info);
}
```

We find the folder of scripts by finding the folder that the application was launched from, and then looking for an enclosed folder named `Script Menu Items`. The routine below finds an enclosed folder id, given a parent folder and a name:

UFinder::GetFolderID

```
long UFinder::GetFolderID(
    FSSpec &inParentFolder,
    Str255 inName)
{
  CInfoPBRec pb;
  DirInfo       *dpb = (DirInfo *)&pb;
  OSErr         err;

  dpb->ioNamePtr = inName ;
  dpb->ioVRefNum = inParentFolder.vRefNum;
  dpb->ioDrDirID = inParentFolder.parID;
  dpb->ioFDirIndex = 0;
  err = PBGetCatInfo(&pb, false);

  // make sure its a folder
  if (err == noErr && dpb->ioFlAttrib & ( 1 << 4))
    return dpb->ioDrDirID;
  else return 0;
}
```

The recipe for iterating through each item in a folder is really ugly. The class below encapsulates the details in an iterator:

StFolderIterator constructor

```
StFolderIterator::StFolderIterator(
    short inVRefNum, long inFolderID)
  : mVRefNum(inVRefNum), mFolderID(inFolderID), mIndex(0)
{
}
```

StFolderIterator:: Next

```
Boolean StFolderIterator::Next(
    HFileParam &ioRec)
{
  ioRec.ioVRefNum = mVRefNum;
  ioRec.ioDirID = mFolderID;
  // reset name field
  if (ioRec.ioNamePtr) ioRec.ioNamePtr[0] = 0;
  ioRec.ioFDirIndex = ++mIndex;
  ioRec.ioResult = noErr;

  PBHGetFInfo((HParmBlkPtr)&ioRec, false);
  return (ioRec.ioResult == noErr);
}
```

Conclusion

Scripting adds very powerful capabilities to your application. The script menu makes it easy for users to attach scripts to a menu in your application. And the code is encapsulated in an `LAttachment`.

UNIFORM RESOURCE LOCATORS

By Jim Straus, URLs@mactech.com

Spare your fingers and find the full list online at: **http://www.mactech.com/URLs.html**

Or (for a limited time), send mail to: **MacTech-URLs@class.com** and you will receive the latest list back.

Latest Updates

Internet Related

Aretha (Frontier)	http://www.hotwired.com/staff/userland/aretha/
Java	http://java.sun.com/
NetPhone	http://www.emagic.com/
Netscape Defrost	http://cygnus.rsabbs.com/~ssykes/nsdefrost.html
Open Door Networks	http://www.opendoor.com/
WebArranger	http://www.cesoft.com/webarranger/webarranger.html
Free World Dialup	http://www.pulver.com/fwd/
Remote Printing FAQ	http://linux1.balliol.ox.ac.uk/fax/faxfaq.html

New Technologies

Linux for PPC	http://liber.stanford.edu/linuxppc/

Other Programmer Resources

Application Generators	http://torgo.astro.ucla.edu/Mac.html

Vendors, Products and Miscellaneous

Altura Software	http://www.altura.com/
CE Software	http://www.cesoft.com/
Computer Literacy	http://www.clbooks.com/
Cyberian Outpost	http://www.cybout.com/cyberian.html
MacCentral	http://www.atcon.com/maccentral/home.html
MacroMedia	http://www.macromedia.com/
Mobilis	http://www.volksware.com/mobilis/
Onyx Technology	http://www.std.com/onyxtech/

Highlights

Internet

CE Software has introduced WebArranger. It is based on the Arrange software (demonstrated at the WWDC several years ago, to much applause) which CE recently acquired. Besides providing a database for your bookmarks, it has the capability to check bookmarks to see if they have changed, and notify you if they have. It also tracks sites you have visited, making a history log, which helps when you have forgotten how you got to some site.

WebArranger **http://www.cesoft.com/webarranger/webarranger.html**

Unless you have been off on some remote island developing the perfect application, you have heard of Java and seen all of the media hype. If you want to find out more about this language, browser technology, elixir of life, killer of Microsoft, go straight to the source.

Java **http://java.sun.com/**

I don't know about you, but I have had Netscape freeze up my Macintosh. Scott Sykes found a problem that appears to be a Macintosh problem, but is actually caused by Netscape. He made an extension to work around the bug. Perhaps it will be fixed by the time you read this, but you might still want to check out his page.

Netscape Defrost **http://cygnus.rsabbs.com/~ssykes/nsdefrost.html**

Free World Dialup is trying an experiment to provide free long distance calling. Based on the idea provided by programs such as VocalTec's Internet Phone, FWD is connecting the Internet back into the normal phone system. Servers are provided that will place calls in their local (i.e. free) calling area. So, if you want to call someone in Paris and there is a server there, you can do so – for free. These guys are looking for some help to develop a Macintosh client. If you like the idea, contact them. This concept is similar to the "Remote Printing" or e-mail-to-fax system that is also running experimentally. Check them both out for some ideas on grass root services through the Internet.

Free World Dialup **http://www.pulver.com/fwd/**

Remote Printing FAQ **http://linux1.balliol.ox.ac.uk/fax/faxfaq.html**

Macintosh

Frank Henriquez has put together a page discussing Application Generators and Application Frameworks, as well as providing links to some available on the Internet. Application generators allow you to draw an interface and then the program generates an application shell for you to fill in. Frameworks are skeletal programs that you fill in with code to do what you want. Sometimes you can use both together. Check out his page and maybe you can simplify your development.

Application Generators **http://torgo.astro.ucla.edu/Mac.html**

If you have ever lacked reading material, or needed to find some technical book, this site is for you. Computer Literacy has one of the largest selections of computer books, reference materials, and anything computer related. They also review new books, have galleys of books not yet in print, and provide access to their book database. A very good reference even if you are not near one of their stores.

Computer Literacy **http://www.clbooks.com/**

Mobilis is an on-line magazine targeted at mobile computer users. It primary deals with PDA class machines, but also deals with issues of interest to users of any mobile computers. It will be especially attractive to you if you are into hand-held computers.

Mobilis **http://www.volksware.com/mobilis/**

Neat Non-Macintosh Site of the Month

Apple and Adobe have sponsored a very hip weekly online magazine called SALON. It looks to be the Internet's answer to the New Yorker, but hipper. The last couple of issues I have read have been great. I could even see it make the reverse jump to a print magazine.

SALON **http://www.salon1999.com/**

Lombard is a brokerage house, but they are providing stock quotes as well. In addition, they are providing historical graphs for any stock of interest. Their one requirement is that you register your name with them (no fee involved).

Lombard **http://www.lombard.com/**

Well, that is it for this month. As always, if you find something interesting, or have updates, send them to URLs@MacTech.com

Thanks this month to Michael D. Crawford, Frank Henriquez, david hirmes, Devon Hubbard, Scott Sykes, Chris Wysocki, Jordan Zimmerman, and many others for their contributions for their suggestions and pointers to new and old sites.

GET YOUR NAME "IN LIGHTS"

Here at *MacTech Magazine*, we rely heavily on outside writers for most of the material that appears in our pages. If readers did not participate in the magazine, sending us their ideas and taking the time to write articles, there would be no *MacTech*. We like to think of *MacTech* as an ongoing dialogue amongst members of the Mac programming world: we facilitate the discussion, but it's the readers who carry it on, by responding to what they read and to their own programming experiences and interests, in writing. Sometimes we know that we need something specific covered, and we approach someone to write an article on that subject; and we do write a few columns in-house each month. But it is reader contributions, in the form of letters, tips, and especially articles, that give the magazine its relevance, its character, and its appeal.

So *MacTech Magazine* is not a staff of writers sending a constant stream of one-way messages outwards; it's a living, evolving network of readers conversing with one another, educating one another, sharing their knowledge, their experience, their interest, their trials and tribulations and joys and successes in the constantly unfolding story of programming the Macintosh. *MacTech Magazine* doesn't just happen: it's what the community makes it. If we carry reports of future trends and technologies, if we teach useful methods, if we review new books and tools, if we provoke thought, provide help, ride the wave of current interests and concerns, it is only because we reflect the thoughts of our readers, who speak through our pages.

You are invited to involve yourself in this exciting conversation amongst readers. You may be working at the cutting edge of programming technology, as part of a heavily funded professional developer effort; you may be a lone hobbyist wrestling to create shareware for the sheer love of it. You may have been programming the Mac since its inception; you may have just switched over from Windows or Unix. You may work in C or C++ or Pascal or AppleScript, from scratch or in a framework. You may write big apps, small apps, custom solutions, extensions, code segments, for profit, for fun, for education, to solve one problem once. No matter who you are, no matter what your credentials may be, if you have a tale to tell, a trick to share, a technique to teach, we want you to consider joining the family of those who write for *MacTech*.

Don't just wait for a topic to be covered in *MacTech!* Don't just wish some technique would be explained better! Take responsibility! Write us an article yourself!

To write for *MacTech*, just send for our Writer's Kit. It's a Microsoft Word file containing the Styles you need to use, and giving lots of helpful advice and information, including all the legal stuff. You can let us know what you're writing about, if you want suggestions or feedback; this is helpful to us, because it lets us plan for the future. Or if you want to, you can just write the article and spring it on us when it's done. If we publish your article, you'll be paid for it!

Write to me, Matt Neuburg, at managing_ed@mactech.com (or one of the other editorial addresses listed on page 2 of the magazine). Ask me for a Writer's Kit! Send me an article! Get published! Make money! See your name in print up there alongside the famous denizens of *MacTech!* And, most important, take the future of *MacTech Magazine* into your own hands!

For Macintosh Programmers & Developers
MacTech™ MAGAZINE

Tips & Tidbits continued from page 88

or the next line should read:

```
GetDItem(theDialog, i + 1, &iType, &iHandle, &iRect);
```

This error is a little hard to detect. Calling `GetDItem` with `itemNo` set to 0 doesn't cause any harm. The only effect this bug has is the final entry in the DITL isn't included in the search. This final item usually isn't a push button, so the author didn't notice the error in his own routine. Otherwise, this code works wonderfully; I use a repaired version all the time!

Mike Trent

A More General Context Sensitive Cursor

In the December issue of MacTech, a technique for making your application WindowShade-savvy was presented, which checked the current content region of the window to see if it was null. A more general method for setting the cursor, which works for WindowShade as well as handling the cases of the cursor outside of the window and when the cursor is in a floating window over the active window is:

```
GetMouse(&mouseLoc);
if (!PtInRgn(mouseLoc,theWindow->visRgn))
    // theWindow is the current window
{
    // Cursor is not in visible region of the window
    // Set cursor to default arrow.
    InitCursor();
}
else
{
    // Adjust the cursor to the proper shape
}
```

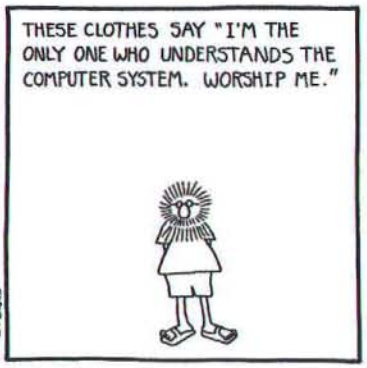
Paul Hyman

Dilbert® *by Scott Adams*

DRESSING FOR SUCCESS

MEN'S BUSINESS CLOTHES

A MAN'S BUSINESS CLOTHES ARE THE MOST IMPORTANT DETERMINANT OF HOW HE IS TREATED.



Reprinted by permission of UFS, Inc.

MacRegistry™

Developer Job Opportunities

If you are a Macintosh developer, you should register with us! We have a database that enables us to let you know about job opportunities. When we are asked to do a search by a client company the database is the first place we go. There is no charge for registering. The database service is free. Geographic Coverage is nationwide.

Marketability Assessment - To get a specific feel for your marketability send a resumé via Email or call. You may also request a Resume Workbook & Career Planner.

Discreet - We are very careful to protect the confidentiality of a currently employed developer.

Scientific Placement is managed by graduate engineers, we enjoy a reputation for competent & professional job placement services and we are Mac fanatics.

1-800-231-5920 • das@spi.com • Fax 1-800-757-9003
http://www.scientific.com

Scientific Placement, Inc.

MT, Box 19949, Houston, TX 77224, 713-496-6100 Fax: 713-496-0373
MT, Box 71, San Ramon, CA 94583 510-733-6168 beth@spica.bdt.com
MT, Box 202676, Austin, TX 78720-2676 512-260-0123 lej@zilker.net
MT, Kenmore Station, Box 15225, Boston, MA 02215 617-424-8372 jen@spbos.pn.com
AppleLink: D1580; Compuserve: 71250,3001; eWorld: spi; AOL: davesmall

---

THE TRATTNER NETWORK

COMPUTER CONSULTING SERVICES

The Trattner Network (TTN) is looking for experienced Macintosh developers for cutting edge opportunities in Northern California and across the country.

TTN represents clients whose projects for consulting and Full Time Employment include development in **OpenDoc, Cyberdog, Copland, Newton, PowerPC, Metrowerks, MacApp,** and many others.

NOW — The Trattner Network has urgent needs for:

- **Software Developers**
- **Hardware/Firmware Engineers**
- **QA/QC Professionals**
- **Project Coordinator/Managers**
- **Multimedia Developers**
- **Network Professionals**

The Trattner Network has a unique history in **Mac consulting** coupled with exposure to emerging technologies. If you are looking for a chance to enhance your skills and marketability, **please send, fax, or link your resume to:**

The Trattner Network
Attn: Emily Hoolhorst
170 State Street, Suite 240 • Los Altos, CA 94022
Phone: 415•949•9555 ext.115 ; Fax: 415•949•1026
AppleLink: trat.net ; E-mail: emily@tratnet.com

THE DIGITAL TALENT SOURCE

MACINTOSH SERVICES

PUBLISH YOUR SOFTWARE FREE

United Software Exchange will market your software nationally at no cost to you.

Interested? For no obligation information email Ralph at **USoftEx@ aol.com** or call **(708) 582-7465**.

USE
UNITED SOFTWARE EXCHANGE

"I can do anything with C++ and a good pair of socks."

– MacXperts Programmer approaching Himalayan village at 20,000 feet.

Macintosh Programmers & Adventurers Wanted

MacXperts is a software development company with immediate openings for Macintosh programmers with C, C++, or other Mac programming experience.

Take the first step to adventure. Call Michael Ruttle at:

1-800-356-8040 or fax / 804-358-3847
Internet: xperts@infi.net
AppleLink: xperts AOL: MacXperts
http://www.macxperts.com/~xperts

MindVision Software

(co–authors of Speed Doubler)

wants to pay you well, insure you and your **family,** give you lots of **vacation** time, great **equipment** to work on, **great people** to work with, listen to **your ideas,** and help get **your name** in the **credits** of some of the most **successful products** in **Mac & PC** history. We ask only that you **try** as **hard** for us as **we will** try for you! If you want to **team up** with a **great** software **company**

MacTech Magazine is your recruitment vehicle

When you need to fill important positions at your company, MacTech Magazine is the consistent choice of companies across the country for hiring the best qualified Macintosh programmers and developers. Let MacTech Magazine deliver your recruitment message to an audience of over 27,000 qualified computer professionals.

Call Ruth Subrin at 805/494-9797

NEWSBITS

By John Kawakami, MacTech Magazine Editorial Assistant

Dylan Pre-release Ships

The Apple Dylan Technology Release is a development environment for the Macintosh platform based on an Object-Oriented Dynamic Language (OODL) called Dylan. The goal of this release is to provide developers with the opportunity to familiarize themselves with Dylan as a language as well as Apple Dylan as a development experience. It is being offered to encourage exploration of the Dylan language and environment.

This is a "Technology Release" because the software is unfinished. It contains a number of bugs, and will not be supported or updated through Apple's standard technical support processes.

Applications under development can be run on any Macintosh and will run native on the PowerPC. The development environment itself runs best on a 68030 or 68040 based Macintosh with at least 20 megabytes of physical memory. For Power Macs, developers should turn off the Modern Memory Manager to run the development environment in emulation. Apple Dylan has been tested on MacOS versions 7.1 and 7.5.

The Apple Dylan Technology Release includes:

- Dylan compiler and runtime
- Integrated development environment featuring incremental development and advanced configurable browsing and viewing of code
- Dylan application framework
- Dylan user-interface builder
- Cross-language support allowing seamless access to existing C code and APIs
- Complete printed documentation

The Apple Dylan Technology Release is hosted on 68K-based Macintosh systems. However, you will be able to produce applications targeting both the 680x0 and Power Macintosh systems.

The development environment lets you create projects with all the advantages of a rapid-prototyping environment. Your project is stored in a database rather than in text files. Customizable browsers let you explore and edit your program from a number of perspectives. For example, you can browse class hierarchies graphically, find all callers of a given function, or find all the objects which reference a given object.

The incremental compiler allows you to change code in a running program and see the results immediately. This gives you freedom as a programmer to explore various options and rapidly improve your software.

The Dylan language is thoroughly object-oriented and contains a number of features which encourage abstraction, leading to cleaner, more maintainable code. Automatic memory management frees you from the burden of manually allocating, tracking, and deallocating memory usage in your application, saving you both programming and debugging time. The language's consistency and familiar syntax ensure that it is easy to learn.

Apple is in active discussions with various partners exploring ways to enhance the technology release in the future. Any future versions of the technology release depend on the successful completion of those discussions.

Orders can be placed to APDA; use order # M4724Z/A. The price is $39.95. To order from North America, please call the appropriate toll-free number:

United States	1 (800) 282-2732
Canada	1 (800) 637-0029

Other international customers may order by contacting one of our licensed resellers in 30 countries, or by calling the following number:

International	(716) 871-6555

Symantec Releases Java DevelopmentTools

Symantec Corporation announced it has licensed the Java Internet programming language technology from Sun Microsystems and released the first Java development environment for Windows 95 and NT, code named "Espresso". Espresso is Symantec's fully integrated development environment which seamlessly incorporates Sun's Java Development Kit (JDK) for the creation of Java applets for Internet web pages. Espresso provides Java developers with class and project management capabilities within a graphical development environment.

Espresso parses the Java Source code on the fly and creates a repository of information about the Java applet and the Java class libraries. This provides a visual representation of the Java application class hierarchy, allowing the user to better understand the standard Java classes, as well as classes of the application. The Class Browser allows developers to browse the Java sources as well as giving them the ability to browse and edit methods, data, and classes. The Class Browser also allows developers to work with the object-oriented pieces of their Java program, as opposed to source files.

Espresso for the Power Macintosh development environment will follow in the first quarter of 1996.

http://sunweb.symantec.com:80/lit/dev/javaindex.html

Language Systems and Spyglass Merge to Form Fortner Research

Language Systems, makers of LS FORTRAN and LS Object Pascal, have merged with Spyglass Inc., makers of advanced data visualization software, to form Fortner Research.

Brand Fortner, Ph.D., the Spyglass co-founder responsible for technical product marketing of its visual data analysis (VDA) tools, is charting the company's course as chairman of Fortner Research. Joining him from Language Systems are Rich Norling, that company's founder, and Guy McCarthy, who retains his titles as president and chief executive officer in the new company. Spyglass, Inc., elected to sell the visualization software to concentrate on developing communications products for the World Wide Web. The Language Systems product line includes LS FORTRAN and LS Object Pascal compilers, and Math 77, a cross-platform library for numerical computation.

The new company will maintain current pricing for both product lines. Fortner Research expects to introduce significant upgrades of existing products in 1996. In addition, it is actively investigating new data technologies emerging from the world of supercomputers. "We're going to deliver new levels of performance and capability that exceed everything we have today," McCarthy predicted.

LS Object Pascal, which recently became a drop-in compiler in the Symantec IDE, will continue to be supported and developed by Fortner Research.

Fortner Research LLC., 100 Carpenter Dr., Sterling, VA 20164. Phone: (703) 478-0181; fax: (703) 689-9535.

http://www.langsys.com/langsys/
info@fortner.com

It Takes Three to Tango (Tango, Butler, WebSTAR)

EveryWare has begun shipping Tango, a full-featured visual development tool that integrates databases with Web servers. Tango lets Web administrators create Web pages that utilize databases without writing any SQL or HTML code.

Tango connects WebSTAR to EveryWare's Butler SQL relational database server. An ODBC version of Butler is slated for release in January of '96. Tango provides the tools allowing webmasters to create:

- electronic shopping malls
- product and pricing catalogs
- chat and conferencing systems
- event registration systems
- enhanced security systems

Tango's approachable visual programming environment increases productivity and allows non-programmers to develop applications for the Web. Webmasters can focus on the graphical and layout elements of the solution while Tango handles the database elements. In addition to the standard editing environment, Tango also has Query Builders, which are like assistants or guides. They allow quick creation of search, insert, update and delete interfaces to the databases.

EveryWare Development Corp., 7145 West Credit Ave. Building 1, Ste. 2, Mississauga, Ontario, Canada L5N 6J7. Phone: (905) 819-1173; fax: (905) 819-1172; FirstClass BBS: (905) 819-9891.

http://www.everyware.com/
info@everyware.com

NetForms Advances Maxum's Quest to Make Your CGIs Obsolete

NetForms is an add-on application which runs on your MacHTTP WWW server. It allows forms entered by users of the server to be automatically converted to formatted HTML documents, which can then be read by other Web clients. In addition to these expected features, NetForms will automatically add links to specified strings (like your name) and automatically manage large lists of pointers to other documents. The price for a single server is $195, and the educational price for a single server is $95.

http://www.maxum.com/NetForms/

Objectivity Announces Objectivity/DB Server for Macintosh

Objectivity, Inc. announced a Macintosh server for Objectivity/DB, the scalable, high-availability, high-performance object database built on a fully distributed architecture. The new Objectivity/DB server for Macintosh incorporates administrative tools with a native Macintosh look and feel, bringing Macintosh peer-to-peer server capabilities in a distributed network of object-oriented information servers.

"Macintosh usability combined with the robustness and scalability of Objectivity's object database will make it easier for developers to build more sophisticated object-oriented applications, such as video-on-demand, for their customers," said Mike Zivkovic, business development manager for Apple Computer's Development Tools Group.

Objectivity/DB supports development language interfaces for C++ and ParcPlace Smalltalk, as well as ANSI-standard SQL with ODBC support for integrating applications with off-the-shelf tools. Objectivity/DB objects are interoperable between language interfaces – for example, an object created with C++ on one platform can be read, updated or deleted using Smalltalk on another platform.

The Objectivity/DB server for Macintosh will be available beginning in the first quarter of 1996. North American pricing for the Objectivity/DB server for Macintosh begins at $155 per single user.

Objectivity. In the US, phone: 415-254-7100; in Europe, phone: +31 1045 83986

http://www.objectivity.com
info@objy.com

MailOrderStore

HOT ITEM! SPECIALS • EXCLUSIVES • HOT ITEMS! NEW!

Learn C on The Macintosh Second Edition By Dave Mark: Learn C on The Macintosh Second Edition: This is a completely revised edition of Learn C on The Macintosh. With this self-teaching, easy-to-understand book and the enclosed CD-Rom, you get everything you need to start programming in this widely used language. New to this edition are updated and enhanced exercises that lead you step by step through the programming fundamentals and C language basics including function, variables, pointers datatypes, data structures, and the file input and output. Also new is completely rewritten code, plus answers and source code for all of the exercises. The new CD-ROM with Metrowerks Code Warrior™ Lite, a special version of one of the hottest Macintosh programming environments, (including a PowerPC version). ~~$34.00~~ **$31.45**

Tricks of The Mac Game Programming Gurus is the ultimate resource for beginning to expert game programmers who already have general programming experience. Complete overview of all the necessary components of game programming on the Macintosh. Hundreds of tips, tricks, and insider secrets from Mac game programming experts on a CD-ROM, packed with valuable tools, utilities, sample code, Code Warrior™ Lite and game demos. Coverage of cutting-edge topics such as QuickDraw 3D and Power Mac optimization and inside info on how Glypha III was created. Unique in the marketplace – no other Mac game programming book is this complete! The book contains instruction, tips, and source code from the top names in Mac game development today. The secrets, examples, and code can't be found anywhere else! These are the tried-and-true tricks that work behind the scenes in the most popular commercial and shareware Mac games. Throughout the book, you'll find special interviews with some of the most well-known Mac game programmers. They reveal their secret solutions created while they developed their popular games. ~~$50.00~~ **$45.00**

OOFILE is the first OODBMS framework to offer a complete solution for application authors. Replaceable backend database, currently Faircom's c-tree Plus for cross-platform royalty-free power. PowerPlant and other frameworks integrated with edit fields, database browsers & more. AppMaker users, generate complete applications. User-friendly syntax makes it easier to migrate from FoxPro and the non-OO world. Demo's on CodeWarrior and AppMaker CD's. $900 for a once-off c-tree bundle, or $1,095 1-year subscription.HTML and character-mode report-writer $195. Full GUI report-writer, including HTML, is $495. Mac Platform Bundle - includes all Mac frameworks, c-tree, and Report-Writer. $1,495

PowerTap™ accelerates software by tapping into multiple processors. Version 3.0 taps into networked Macs and all processors found in the new multi-processor Macs. Developers can speed up their applications without having to learn about networking, communications and task scheduling algorithms. The PowerTap™ library has the easiest API - it behaves as a simple black box where tasks are submitted and results retrieved. Full error recovery is built-in, so your job will complete no matter what. PowerTap's advanced scheduling algorithms ensure optimal assignments and the fastest execution possible. It is compatible with all Macintosh hardware, software and major compilers. Version 1 comes with 2 remotes - $1200 Version 2 comes with 5 remotes -$1900 Version 3 comes with unlimited remotes - $2700

Roaster DR1™– Be the first Macintosh developer on your block to take advantage of the unique capabilities of Sun's™ new Java™ Programming Language! Developer Release 1 of Roaster™, new from Natural Intelligence, Inc., is the first ever development environment for writing, testing, and running Java™ applets on the Macintosh. Features include: fully integrated development environment – project window that includes a Finder-like view of packages – lightning-fast – Macintosh native compiler – source code editor with powerful search features and intuitive use interface – runtime engine for quick and easy applet testing. When you purchase Roaster™, you are entitled to: unlimited, personalized tech support throughout the length of your subscription DR 1, DR 2, Roaster 1.0, and one additional update. Roaster™ is accelerated for Power Macintosh. Requirements Macintosh or Macintosh compatible computer with a Motorola 68020 or higher or Power PC processor; CFM-68K (for 68K machines); 8MB RAM; Color QuickDraw; System 7.1.2 or later; (System 7.5 or later for 68K machines); CD-ROM drive to install the software. Price: $399 Special Price: $299

Symantec C++ is the industry-standard Macintosh development system--and now it's native for Power Mac. You can develop full-featured Power Macintosh applications quickly and easily using revolutionary new features that save time and enhance your productivity throughout the development cycle. Environment includes: A true native Power Mac implementation of the C++ language, including support for templates and multiple inheritance; MrC/MrC++ compilers for fast Power Mac executable code (22% faster than with the standard Symantec or Metrowerks compilers); Visual Architect for fast, easy GUI generation; A new THINK project management system that supports large, complex applications, nested projects, and hierarchical project organization; A new editor/browser that displays classes and automates many editing functions; A powerful, easy-to-use source code debugger; The industry-standard THINK Class Library; Free subscription—receive the next two updates

MACTECH MAGAZINE PRODUCTS & ORDER INFORMATION

E-mail, Fax, write, or call us. You may use your VISA, MasterCard or American Express; or you may send check or money order (in US funds only): MacTech Magazine, P.O. Box 5200, Westlake Village, CA 91359-5200, Voice: 805/494-9797, Fax: 805/494-9798

If you are an e-mail user, you can place orders or contact customer service at:

- AppleLink: MT.CUSTSVC
- CompuServe: 71333,1063
- Internet: custservice@xplain.com
- GEnie: MACTECHMAG

MACTECH WEB™ SITE:
http://www.mactech.com
For complete product info

SUBSCRIPTIONS
US Magazine: $47 for 12 issues
Canadian: $59 for 12 issues
International: $97 for 12 issues
US Magazine with Source Code Disk: $124 for 12 issues
Canadian Magazine w/Source Code Disk: $136 for 12 issues
International Magazine w/Source Code Disk: $194 for 12 issues

CD-ROM
MacTech CD-ROM, Volumes 1-10: Includes over 1230 articles from all 115 issues (1984-1994) of MacTech Magazine (formerly MacTutor). All article text and source code. Now in THINK Reference format. The CD includes Symantec's THINK™ Reference 2.0, working applications with full documentation, product demos for developers and more. See advertisement, this issue: $49. Upgrades $39. E-mail, call or write for info.

BOOKS
The Best of MacTutor, Vol. 1: Sold Out
The Complete MacTutor, Vol. 2: Sold Out
The Essential MacTutor, Vol. 3: $19.95
The Definitive MacTutor, Vol. 4: $24.95
The Best of MacTutor, Vol. 5: $34.95
Best of MacTutor Collection, Volumes 3 – 5: $69
Best of MacTutor, Volumes 6, 7, 8 & 9: Not available

DISKS
Source Code Disks: $8 each
Topical Index (1984-1991) on disk: $5

MAGAZINE BACK ISSUES
Volumes 3, 4, 5, 6, 7, 8, 9 and 10: $5 each (subject to availability)

California residents include 8.25% sales tax on all software, disks and books.

Allow up to 2 weeks for standard domestic orders, more time for international orders.

PLEASE NOTE
Source code disks and journals from MacTech Magazine are licensed to the purchaser for private use only and are not to be copied for commercial gain. However, the code contained therein may be included, if properly acknowledged, in commercial products at no additional charge. All prices are subject to change without notice.

free when you send in your registration card and more! Price: $399

Symantec C++ for 68k, allows you to build applications faster and easier with the powerful combination of fully integrated visual tools and the latest in C and C++ compiler technology. C++, now the standard in development languages, provides an object-oriented approach to application development, and the resulting code is extensible, reliable, and maintainable. Includes: Integrated Environment with full source-code debugging, integrated editing and browsing, and resource creation and editing; support for standard language features such as templates and multiple inheritance as well as quick compile times and highly optimized code; Incremental Linker which eliminates long link times; THINK Class Library – a mature, C++ based, application framework provides a solid foundation on which to build factored and scriptable applications with support for C++ exceptions, run-time type identification, persistent objects, and AppleEvents; Visual Architect™ to visually design your user interface and automatically generate complete THINK Class Library source code; THINK Inspector which provides you with quick and easy navigation of your application's class hierarchy while it is running; Project Models to start new projects quickly from templates either provided in this package or created to fit your own needs; Source-Code Control with integration with Apple's SourceServer (included) provides reliable version control and supports team programming; Support for Scripting lets you automate complex, multiproject operations using AppleScript or Frontier scripting; Powerful Standard Libraries which includes IO Streams, ANSI standard C library, and sample programs. Full source code is included; Extensibility supports third-party editors, and AppleScript and MPW tools (available from Apple) and allows access to THINK Reference; Create applications; desk accessories; device drivers; and any kind of code resource, including HyperCard XCMDs and XFCNs, system extensions, and control panels; Migration Path to Power Mac – Create applications that run on 68K Macs and Power Macs (emulated). These applications can easily be migrated to native Power Mac, by trading up to Symantec C++ for Power Mac. Price: $79

Mastering the THINK Class Library by Richard Parker. Now that Symantec's long-awaited PowerPC native compiler is here, developers are taking another look at THINK. This book provides a thorough examination of Symantec's extensive Class Library and the Visual Architect, a graphic user interface development tool that allows you to produce commercial-quality applications with a minimum of effort. A complete description of the structure and operation of the TCL includes explanations of all code generated by the Visual Architect, any necessary custom code, and the operation of this code. Visual Architect tutorials provide you with a step-by-step approach for simplifying the development of complex Macintosh applications. 496 pages ~~$29.95~~ **$26.96**

A Fragment of Your Imagination by Joe Zobkiw. Here's some practical help for creating code resources and code fragments for the Macintosh and Power Macintosh. Rather than simply gathering and indexing chunks of this vital code, the author provides thorough explanations to teach you more about how the Macintosh system functions as a whole. He also provides hard to find information about techniques used to structure and build fat, safe fat, and accelerated code resources for use on both 680x0 and Power Macintosh. All code is reusable and is provided on the disc, along with Metrowerks Code Warrior Lite. Book/CD-ROM, 528 pages ~~$39.95~~ **$35.96**

Inside CodeWarrior 8: Includes CodeWarrior IDE User's Guide. This manual shows you how to use the CodeWarrior IDE (Integrated Development Environment). It shows you how to create software for 68K and PowerPC Mac OS, Win32/x86, and Magic Cap. It also shows you how to use ToolServer from the IDE and how to control the IDE using AppleScript. The next highlight is CW Error Messages, which describes the errors you might encounter while using CodeWarrior compilers and linkers. This manual contains descriptions, source code examples, and fixes to these errors, as well as the Debugger Manual, which was updated for CW7 including new inline and exceptions debugging, and a new troubleshooting section. Next up is the MPW Tools Manual, which shows you how to use Metrowerks compilers, linkers, and other tools under the MPW Shell. Updated for CW7, new chapters comparing Metrowerks and other MPW tools, and then onto C, C++, and Assembly, and Pascal Language Reference, covering the Metrowerks implementation of C, C++, and 680x0 assembly language programming, updated for CW7. The Pascal refs also include the new UNSIGNEDWORD and UNSIGNEDLONG. Also included for this time are Profiler and ZoneRanger Manuals updated for CW8. $34.95

Inside PowerPlant is the Power-Plant Manual, and contains information about creating PowerPlant applications using the CodeWarrior IDE and PowerPlant Constructor, and describes major PowerPlant classes and resources. Also included are the PowerPlant Constructor Manual, including View, TextTraits and Custom Types editing, and PowerPlant Library Reference, covering all classes and functions in PowerPlant, updated for CW8. $34.95

BBEdit 3.5 from Bare Bones Software is now better than ever. In addition to being Accelerated for Power Macintosh, this powerful, intuitive text editor offers integrated support for THINK C 7.0, Metrowerks CodeWarrior 6, THINK Reference 2.0 and MPW ToolServer. Version 3.1 adds even more capability, including "soft" wrapping of text on screen and numerous refinements and improvements to the user interface. BBEdit's many features include: Integrated PopupFuncs™ technology for speedy navigation of source code files (C, C++, Pascal, Rez, 68K Assembler, and Fortran), unique 'Find Differences' command (BBEdit can find differences between projects and folders as well as files), support for Macintosh Drag and Drop for editing and other common tasks, PowerTalk support for reading, sending and composition of PowerTalk mail, scripting via any OSA compatible scripting language including AppleScript and Frontier 3.0, and fast search and replace with optional "grep" matching and multi-file searching. BBEdit's robust feature set and proven performance and reliability make it the editor of choice for professionals and hobbyists alike. $99

New Low Price!

QC™ by Onyx Technology, is a system extension that stress tests code during runtime for common and not-so-common errors. Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking. QC is extremely user friendly for the non-technical tester yet offers an API for programmers who want precise control over testing. QC is Also available in Japanese. $99.95

FaceSpan™ v2 is an extensible Rapid Application Designer (RAD) that makes building applications quick and easy. It combines an interactive, visual interface design environment with the object-oriented power of AppleScript or any OSA language. Best of all, FaceSpan allows you to integrate the capability of scriptable programs into your custom application. Your FaceSpan applications can include any number of windows, dialogs, palettes, and menus. In them, you can display scrolling lists, popup menus, scrolling text, movies, multi-column tables, pictures, icons, buttons, and others. While no scripting is needed for standard behaviors, every item may have its own script. You can even program custom objects using Pascal or C. Try the perfect choice for MIS professionals, power users, consultants, and programmers. They ♥ FaceSpan! Includes a royalty-free distribution license, for unlimited runtime users, of your FaceSpan-based applications. Also included is a FREE UPGRADE to the next version for registered users. $199

Scripter® The Authoring and Development Environment for AppleScript™. Scripter, the Script Construction Set, is the foremost, comprehensive tool for creating and debugging AppleScript scripts. Scripter is a shortcut to Applescript's full capabilities, is both powerful and easy to use, and appeals equally to novices and experts. Scripter offers the largest collection of tools to answer the needs of every AppleScript user, containing over 35 features, including: **Superior vocabulary access** – point-and-click assembly of commands and object specifications; command window for experimentation. **Shortcuts and extended editing capabilities** – extensive drag-and-drop, six-function find-and-replace; navigation markers; script library collection facility; many other timesavers for faster scripting. **Interactive debugging** – comprehensive variable watcher, expression evaluation, enhanced trace log, and real single step debugging! Other features include: integration with FaceSpan and background processing. Unlike other scripting tools, which are either based on the original Script Editor concept, or are designed to look more like traditional programming tools, the designers of Scripter understood from the outset that scripting is different from writing C code. Scripter will change the way you work with AppleScript. From expert script design to user-friendly editing and implementation, Scripter is the natural companion to AppleScript for all levels of proficiency. ~~$199~~ **$179**

New Low Price!

Macintosh Programmer's Toolbox Assistant CD-ROM – Instant electronic access to Inside Macintosh essentials. Now Macintosh programmers can get quick access to over 4,000 Toolbox calls that are at the heart of Macintosh system software. The definitions of these data structures, resources, constants, and functions are documented in the Inside Macintosh series and are essential information for anyone developing Macintosh software. Macintosh Programmer's Toolbox Assistant is a CD-ROM that harnesses the power of one of the best search and viewing engines in the industry. It allows programmers to access the Toolbox calls quickly from their development environment. With hypertext links allowing programmers to view related topics easily. Macintosh Programmer's Toolbox Assistant is the ultimate electronic reference tool for Macintosh programmers. **$99.95**

Inside Macintosh®: CD-ROM by Apple Computer, Inc. Inside Macintosh is the essential reference for programmers, designers, and engineers for creating applications for the Macintosh family of

SPECIALS • EXCLUSIVES • HOT ITEMS!

computers. Inside Macintosh CD-ROM collects more than 25 volumes in electronic form, including: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components. Now programmers will be able to access over 16,000 pages of the information they need directly from their computers. Hypertext linking and extensive cross referencing across volumes allows programmers to search and explore this library in ways that are unique to the electronic medium. Every Macintosh programmer will regard Inside Macintosh CD-ROM as their most important resource. **$99.95**

ScriptWizard™ 1.5 is the latest version of the best-selling script-editing and debugging tool that combines the power of a professional development environment with the ease of use that you expect of Macintosh™ software. Compatible with all Apple® Open Scripting Architecture languages, including AppleScript™, ScriptWizard improves your productivity by delivering testing and debugging facilities that are as intuitive as they are powerful. ScriptWizard makes life easier for scripters by emphasizing features that speed script development. Some of the most significant enhancements to scripter productivity include the ability to single-step scripts (now allowing true statement-level stepping), watch variable values as scripts execute, jump instantly to frequently used places in a script and find and replace specific text. Full drag and drop text editing is supported. ScriptWizard delivers an intuitive development, testing and debugging environment for rapid script creation with essential tools up-front for easy access. **$89**

CodeWarrior™ 8 CD by Metrowerks comes in two versions – Bronze and Gold. These CDs contain the CodeWarrior 7 development environment including C++, C and Pascal compilers; high-speed linkers; native-mode interactive debuggers; and a powerful new application framework called PowerPlant for rapid Macintosh development in C++. Bronze generates 680x0 code. Gold generates both 680x0 and PowerPC code. Comes in two versions – Bronze and Gold. New to these versions is an Integrated Class Browser. Faster code, Better Code Generation for PowerPC. Updated OpenDoc™ support. New Networking Classes in PowerPlant. New Editor Functions. Zero Overhead Exceptions for C++. Libraries for the new Be Operating System. CodeWarrior 8 Gold. With CodeWarrior's plug in architecture, choose your target platform from within one Integrated Development Environment. Gold supports the following platforms: PowerPC™: Mac™OS, PowerTV™, Be™OS for BeBox™. 68K: Mac™OS, Magic Cap™. x86: Windows 95™, Windows NT™. It includes C/C++ and Object Pascal. CodeWarrior 8 Bronze. Macintosh programming solution that generates code that runs native on 68K and runs emulated on PowerPC. Supports C/C++ and Object Pascal. Gold $399. Includes a 1 year MacTech subsciption. Bronze $149. Includes a 6-month subsciption.

BOOKS

10% OFF ALL BOOKS!

Learn C on The Macintosh Second Edition By Dave Mark: See page 76

Tricks of The Mac Game Programming Gurus See page 76

Macintosh C Programming Primer Volume I, Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed. This new edition of this Macintosh programming bestseller is updated to include recent changes in Macintosh technology, including System 7, new versions of THINK C and ResEdit, and new Macintosh machines. Readers will learn how to use the resources, Macintosh Toolbox and interface to create stand-alone applications. 672 pages, ~~$26.95~~ **$24.25**

Macintosh C Programming Primer Volume II, Mastering the Toolbox Using THINK C by Dave Mark. Volume II picks up where Volume I leaves off, covering more advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager: 528 pgs. ~~$26.95~~ **$24.25**

Macintosh OLE2 Programmer's Reference: Working With Objects: Provides a complete reference to the extensible protocol of Object Linking and Embedding, version 2.01 for Macintosh System 7. Understanding of C/C++ helpful, but not necessary, and comes with a CD. Working With Objects describes the visual and interactice interfaces that support the component objects, provides details of the OLE 2.01 for the Macintosh user Interface, addresses the issues of object class registration, shows how to implement the drag and drop objects from one application to another, covers the interface that exposes the basic embedding functionality, and includes descriptions of API functions. ~~$44.95~~ **$40.45**

Macintosh Pascal Programming Primer Volume I, Inside the Toolbox Using THINK Pascal by Dave Mark and Cartwright Reed. This tutorial shows programmers new to the Macintosh how to use the Toolbox, resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages ~~$26.95~~ **$24.25**

Learn C++ on the Macintosh by Dave Mark. After a brief refresher course in C, Learn C++ introduces the basic syntax of C++ and object programming. Then you'll learn how to write, edit, and compile your first C++ programs through a series of programming projects that build on one another as new concepts are introduced. Key C++ concepts such as derived classes, operator overloading, and iostream functions are all covered in Dave's easy-to-follow approach. Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages, ~~$36.95~~ **$33.26**

Programming Primer For The Macintosh® Volume 1 by John Whittle and Judy May. This book provides an introduction to Macintosh programming, using C++ as the example language, and provides realistic, easy to follow, programming examples designed to work with either Symantec® C++ or Metrowerks® CodeWarrior™. Also includes one 3.5" disk with source code for the programming examples, along with numerous, useful, public domain utilities to use with each compiler. ~~$37.95~~ **$34.15**

Mastering the THINK Class Library by Richard Parker. See page 68

Programming in Symantec C++ for the Macintosh by Judy May and John Whittle. This book will introduce you to object-oriented programming, the C++ language, and of course Symantec C++ for the Macintosh. You don't have to be a programmer, or even know anything about programming to benefit from this book. Programming in Symantec C++ for the Macintosh covers everything from the basics to advanced features of Symantec C++. If you are a Think C or Zortech C++ programmer who wants to learn more about object-oriented programming or what's different about Symantec C++, there are chapters specifically for you. Includes helpful examples of C++ code that illustrate object-oriented programs. ~~$29.95~~ **$26.95**

Symantec C++ Programming for the Macintosh, Second Edition by Neil Rhodes & Julie McKeehan is the perfect introduction to C++ programming. ~~$45.00~~ **$40.50**

Teach Yourself Mac C++ Programming in 21 Days by Namir Clement Shammas is the easy-to-follow 21-day format teaches readers how to program in C++ using the Symantec C++ compiler. ~~$29.99~~ **$26.99**

Writing Localizable Software for the Macintosh by Daniel R. Carter. 469 pages. ~~$26.95~~ **$24.25**

Global Interface Design, A Guide to Designing International User Interfaces by Tony Fernandes, AP Professional. Global Interface Design addresses the issues involved in product development for a global market with a "real world" focus. While covering major areas developers should address during the development cycle, Tony Fernandes provides insight into researching cultural differences. This book examines the differences found all over the world, such as cultural symbolism and taboos, and how they impact user interfaces. ~~$34.95~~ **$32.35**

Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++ ~~$19.50~~ **$17.55** Call for more information.

Software By Design: Creating User Friendly Software by Penny Bauersfeld (Series Editor: Tony Meadow). This excellent reference provides readers with a thorough how-to for designing software that is easy to learn, comfortable to operate and that inspires user confidence. Written from the perspective of Macintosh, but compatible with all platforms. Stresses user input from initial design, through prototyping, testing and revision. Provides tools for analyzing user needs and test responses. Includes exercises for sharpening user-oriented design skills. ~~$29.95~~ **$26.95**

Macintosh Programming Techniques by Dan Sydow (Series Editor: Tony Meadow). This tutorial and handbook provides a thorough foundation in the special techniques of Macintosh programming for experienced Macintosh programmers as well as those making the transition from DOS, Windows, VAX or UNIX. Emphasizes programming techniques over syntax for better code, regardless of language. Guides the reader through Macintosh memory management, QuickDraw, events and more, using sample program in C++. Disk includes an

BOOKS

10% OFF ALL BOOKS!

interactive tutorial, plus reusable C++ code. ~~$34.95~~ **$31.95**

More Mac Programming Techniques: More Mac Programming Techniques goes beyond the fundamentals of Macintosh programming With this hands-on guide and tutorial, you'll expand on the basic foundation of programming to develop truly powerful applications. Inside you will take a detailed look at the units of a Macintosh program – from the INITs to custom controls. Along the way, you'll learn solid techniques you can apply anywhere, including tricks and techniques. You will learn file resources from the ground up, and how to build custom menus, add custom controls, including buttons, and slider controls, MDEF, and CDEF resources. The book will also help you to handle INITs, making preference files, and how to print flawlessly from your programs. ~~$39.95~~ **$35.95**

Macworld Ultimate Mac Programming by Dave Mark. Reveals the secrets of Mac programming and presents important, timesaving techniques. ~~$39.95~~ **$35.95**

Mac Screamer, The Ultimate Macintosh® Supercharging Kit by Jan Harrington covers 30 Macintosh models, including the Classics, LCs, PowerBooks, and Quadras and gives software solutions and hardware tips to accelerate Mac performance. It lets readers in on do-it-yourself tips that can save them over 25% on upgrade costs. ~~$35.00~~ **$31.50**

Programming for System 7 by Gary Little and Tim Swihart, is a hands-on guide to creating applications for System 7. It describes the new features and functions of the operating system in detail. Topics covered include file operations, cooperative multitasking, Balloon Help, Apple events, and the File Manager. Numerous working C code examples show programmers how to take advantage of each of these features and use them in developing their applications. 384 pages. ~~$26.95~~ **$24.25**

Guide to Macintosh System 7.5 by Don Crabb. ~~$25.00~~ **$22.50**

A Fragment of Your Imagination by Joe Zobkiw. See page 77

How To Write Macintosh Software by Scott Knaster is a great source for understanding Macintosh programming techniques. Drawing from his years of experience working with programmers, Scott explains the mysteries and myths of Macintosh programming with wit and humor. The third edition, fully revised and updated, covers System 7 and 32-bit developments, and explores such topics as how and where things are stored in memory; what things in memory can be moved around and when they may be moved; how to debug your applications with MacsBug; how to examine your program's code to learn precisely what's going on when it runs. 448 pgs. ~~$28.95~~ **$26.05**

Danny Goodman's Macintosh® Handbook Featuring System 7 by Danny Goodman with Richard Saul Wurman. It includes over 100 spreads break down and clarify Mac problems and includes insider's tips. ~~$29.95~~ **$26.95**

Real World Apple Guide, For The Mac is the much anticipated help and navigational aid component of the new Apple System 7.5 OS. The book is a practical introduction to Apple Guide for programmers. It explains the design and function of Apple Guide, how to design your own guides using Apple Script. Comes with a disk of sample Apple Guides for Apple Guide-compliant applications. ~~$39.95~~ **$35.95.**

Danny Goodman's Apple Guide Starter Kit by Danny Goodman and Jeremy Joan Hewes. Two highly respected experts offer a different approach for creating your own Apple Guide databases. With Danny's Guide Starter program you can make guides quickly and easily, without having to learn a scripting language, write coded files, or use several different files and programs to produce your database (which is what you'd have to do without the program). The authors provide advice and tips on how to design a good Guide, from planning and creation through testing, revising, and indexing. Book/disk, 320 pages. ~~$34.95~~ **$31.46**

HyperTalk® 2.2: The Book Second Edition by Dan Winkler, Scott Kamins, and Jeanne DeVoto is the most complete, authoritative source on HyperTalk 2.2 programming and troubleshooting. It covers each language element of HyperTalk 2.2 (including the odd quirk or bug). ~~$35.00~~ **$31.50**

The Complete HyperCard® 2.2 Handbook Fourth Edition by Danny Goodman is the biggest-selling Mac book – newly revised and updated for version 2.2. It shows how to build working applications using the latest version of HyperCard and covers text, painting tools, extension commands (XCMDs), scripting in HyperTalk, and more. ~~$35.00~~ **$31.50**

Dan Shafer Presents the Power of Prograph CPX is a hands-on, project-centered approach to learning the most revolutionary object-oriented programming language on the planet. The language Kurt Schmucker likes best. The language that programmers actually report having "fun" programming. This 550-page book takes you step by step through three interrelated projects of increasing complexity. Along the way you'll learn the underlying Prograph language, how to use the power of lists, and the important aspects of the CPX classes and object editors. Includes disk with all code in the book. ~~$49.95~~ **$44.95**

Visual Programming With Prograph CPX by Scott B. Steinman and Kevin G. Carver. This is the first book on Prograph CPX available through the book trade. It covers the only commercially supported visual programming language at a time when many programmers and managers, faced with continuing productivity problems, are searching for better programming environments. Prograph CPX is much more than such GUI-enhanced traditional languages as Visual Basic: It literally allows you to draw your program flow using icons and create a complete application without writing a line of code. This book is an introduction to the language and a guide for advanced users, for both Macintosh and Windows-based machines. Prograph is a fully pictorial, general-purpose, object-oriented language that speeds development with an integrated environment for design, coding, testing and debugging; with its OO framework for sophisticated GUI development; with its support for calls from C, C++, Pascal, and other routines; with its DAL, ORACLE, Sybase, AS/400 client/server DB support; and many other powerful features. ~~$34.00~~ **$30.60**

Graphic Gems V Edited by Alan W. Paeth is the newest volume in The Graphic Gems Series. It is intended to provide the graphics community with a set of practical tools for implementing new ideas and techniques, and to offer working solutions to real programming problems. These tools are written by a wide variety of graphics programmers from industry, academia, and research. The books in this series have become essential, time-saving tools for many programmers. It is the latest collection of graphics tips in The Graphic Gems Series written by the leading programmers in the field. It contains about 50 new gems displaying the most recent and innovative techniques in graphics programming. Also included is new gems in ellipses, splines, Bezier curves, and ray tracing. Includes a disk which contains source code from all five volumes and is available in both IBM and Macintosh versions. CONTENTS: Algebra and Arithmetic. Computational Geometry. Modeling and Transformation. Curves and Surfaces. Ray Tracing and Radiosity. Halftoning and Image Processing. Utilities. ~~$49.95~~ **$44.95**

Applied Mac Scripting Applied Mac Scripting covers AppleScript™, Frontier, QuicKeys, Tempo II, nShell, FaceSpan Application Builder, Scripting PlainTalk and System 7.5. With this hands on tutorial Tom Trinko shows you how to automate your Macintosh activities by learning how to use the AppleScript and Frontier scripting environments. You will learn the overall approach to designing and developing powerful scripts, and to harness the capabilities of a wide variety of Macintosh applications into the integrated productivity tools. This includes such things as the newspaper script which combines the power of SITcomm, MacWrite Pro, and Filemaker Pro, or QuarkXpress. Whether you are a power user or experienced Mac Programmer you will learn valuable new techniques for Mac automation. ~~$34.95~~ **$31.45**

Danny Goodman's AppleScript Handbook Second Edition by Danny Goodman is a self-contained kit shows the reader how to customize and extend the capabilities of any Macintosh computer – no programming experience needed! This enhanced and expanded edition of The Complete AppleScript Handbook focuses on putting AppleScript to work in all sorts of practical situations. In addition, Danny shows you how to apply the same principles to other popular scripting systems, such as UserLand Frontier and QuicKeys. Shows readers how to use scripts to enhance the Macintosh environment, automate many processes, link data between applications, and much more. This book provides a wealth of all-new examples showing how to integrate AppleScript with the Finder, spreadsheets, desktop publishing programs, graphics applications, databases, telecommunications programs, utilities, and HyperCard. The accompanying 3 1/2" disk is jam-packed with over $100 worth of software, including AppleScript 1.1, valuable utilities, and powerful, ready-to-use scripts. ~~$39.00~~ **$35.00**

The Complete AppleScript® Handbook by Danny Goodman is a self-contained kit to customizing and enhancing the Macintosh environment. The disk contains AppleScript 1.1 Runtime, Chang Labs TableServer, and useful, ready-to-run scripts. It also shows the Mac user how to automate many processes – no programming experience necessary. ~~$35.00~~ **$31.50**

The Tao of AppleScript: BMUG's Guide to Macintosh Scripting, Second Edition by Derrick Schneider & Hans Hansen. This updated bestseller is a complete, natural introduction to AppleScript programming essentials. Readers learn how to customize applications, automate tedious tasks, and create programs without having to use a complex programming language. 2 disks contain AppleScript, QuickTime, StuffIt Lite, ResMover, and other helpful utilities. Progressive structure meets the needs of any Mac user, regardless of experience. Professional instructions are mixed with practical examples for easy learning ~~$29.95~~ **$26.95**

BOOKS

Wireless For The Newton Software Development for Mobile Communications by Julie McKeehan and Neil Rhodes is a book that picks up where Programming for the Newton left off, teaching the reader how to develop Newton® software on the Macintosh. The enclosed floppy disk provides a sample application, as well as a fully functional demonstration version of Newton Toolkit™ (NTK™), Apple Computer's complete development environment for the Newton®. Gives hands-on Newton environment training with sample code created specifically for the Newton®. The authors are external faculty at Apple Developer University teaching classes on programming for the Newton®. Programming experience is assumed, although not in any particular language. Enclosed is a floppy disk which contains source code for a Newton application, as well as demonstration NTK™. ~~$34.95~~ **$31.45**

HOT ITEM!

Basic For The Newton, Programming for the Newton Using NS Basic by John Schettino & Liz O'Hara. This book shows owners of Newton devices how to become Newton programmers using BASIC. The authors use a straight-forward "programming by example" approach, which should have you writing your own Newton programs right away. It includes one 3.5" disk containing Demonstration NS BASIC and over fifty example programs from the book. It is Multi-platform in that teaches programming for the Newton using a Macintosh, a Windows-based PC, or on the Newton device itself. ~~$35.95~~ **$32.35**

Programming for the Newton Software Development with NewtonScript by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith. Programming for the Newton: Software Development with NewtonScript is an indispensable tool for Newton programmers. Readers will learn how to develop software for the Newton on the Macintosh from people that developed the course on programming the Newton for Apple Computer. The enclosed 3.5" disk contains a sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK), Apple Computers complete development environment for the Newtons. A Publication of AP Professional May 1994, Paperback, 393 pp. ~~$29.95~~ **$26.95**

Metrowerks CodeWarrior Programming by Dan Parks Sydow. Includes CodeWarrior Lite, and Full Coverage of PowerPlant™. The best information on Metrowerks CodeWarrior 6, giving full coverage to the Gold Edition. Even if you don't already own CodeWarrior 6, you'll still be able to work with the examples in this book, using the CodeWarrior 6 Lite CD that comes with it. ~~$39.95~~ **$35.95**

HOT ITEM!

C++ Programming With CodeWarrior Beginning OOP for the Macintosh and Power Macintosh by Jan L. Harrington from AP Professional. This book shows programming novices object-oriented programming techniques for the Macintosh, Power Macintosh, and Mac OS compatibles, using C++ as the example language and Metrowerks and CodeWarrior as the example compiler. The enclosed CD-ROM contains example code from the book and a full-function Metrowerks CodeWarrior compiler for running these examples. ~~$35.95~~ **$32.35**

HOT ITEM!

Optimizing PowerPC Code: Programming the PowerPC in Assembly Language – To take full advantage of the potential of the PowerPC, Developers need to master the Assembly Language techniques. This book shows how to use the Assembly Language in PowerPC Programs to produce faster more robust software. ~~$39.95~~ **$35.96**

HOT ITEM!

Inside CodeWarrior 8: See page 77

Inside PowerPlant: See page 77

Power Macintosh Programming Starter Kit by Tom Thompson. This is the first tutorial/reference for programmers who want to enter the new world of the PowerPC chips. Users find all the details on the new microprocessors, the new RISC architecture, and how to write native code and emulation operations to create their own software for the Macintosh PowerPC. CD-ROM includes a unique compiler for writing code easily. The all-in-one book that gets programmers the information and tools they need. Programming examples reinforce explanations of code and programming tools ~~$39.00~~ **$35.10**

The ResEdit All Night Diner by David Ciskowski. An idea-filled menu and introduction to the joys of customizing software – and adding personality to the Mac with ResEdit! Shows readers how to customize default icons, the text of menus and dialog boxes, cursors, pointers, and more. Provides specific recipes for doing creative things with ResEdit – plus how to avoid problems. Disk features ResEdit program, plus lots of sample resources ~~$24.95~~ **$22.45**

ResEdit™ Complete, Second Edition by Peter Alley and Carolyn Strange. With ResEdit, Macintosh programmers can customize every aspect of their interface form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package. ~~$34.95~~ **$31.45**

Sad Macs, Bombs, Disasters and What to Do About Them by Ted Landau comes to the rescue with your Macintosh problems. From fractious fonts to the ominous Sad Macintosh icon, this emergency handbook covers the whole range of Macintosh problems: symptoms, causes, and what you can do to solve them. 640 Pages. ~~$24.95~~ **$22.45**

Macintosh® Crash Course by Glenn Brown shows Macintosh power users what to do when things go wrong with their system. Macintosh Crash Course shows readers how to overcome Macintosh system crashes, system lock-ups, and various, frustrating and cryptic error messages they regularly encounter. It includes a CD-ROM with shareware and freeware to help the user diagnose and repair system failures. Includes up-to date coverage through Macintosh System 7.5, Managing memory, Hardware diagnostics, File recovery, PowerBook problems, PowerPC problems, network utilities, hard drive repair utilities, SCSI problems, conflicts and solutions and File synchronization and utilities. ~~$29.95~~ **$26.95**

Multimedia Authoring: Building and Developing Documents by Scott Fisher addresses the concerns that face anyone trying to create multimedia documents. It offers specific advice on when to use different kinds of information architecture, discusses the human-factors concepts that determine how readers use and retain information, and them applies these findings to multimedia documents, covering the high-level issues concerning planners and authors of multimedia documents as well as those involved in evaluating or purchasing multimedia platforms. Includes one 3.5" high-density disk. ~~$34.95~~ **$31.45**

Multimedia Starter Kit for Macintosh by Michael D. Murie. This hands-on book offers the latest and greatest in multimedia for the Mac! Readers learn how to design their own multimedia projects step by step, then try it themselves with the demos, graphics, clips, and sample projects on the CD-ROM! CD-ROM contains QuickTime, sound and graphics clips and utilities, sample projects, and more. How to choose and use a variety of Macintosh multimedia tools and presentation environments. Includes demos of Adobe Illustrator, Premiere, Heizer Software programs, and more ~~$39.00~~ **$27.00**

QuickTime Starter Kit for Macintosh by Robert A. Lettieri & Judith Stern. This is the ultimate package for getting productive and having fun with Macintosh movie-making. Easy steps and valuable software help readers play, make, and edit QuickTime movies. CD-ROM includes QuickTime tools, movie clips, shareware, and demos of Premiere and other programs. Written by members of the respected Berkeley Macintosh User Group. Tips on the best ways to bring live-action video to Mac multimedia ~~$45.00~~ **$40.50**

3-D Starter Kit for Macintosh by Sean Wagstaff. The complete reference to 3-D graphics on the Macintosh – ideal for beginning to intermediate product designers, illustrators, graphic designers, multimedia developers, animators, and video producers, as well as architects and engineers! Covers more than 50 major Macintosh 3-D imaging software packages – the most comprehensive book available. Lots of idea-packed examples that illustrate how 3-D products work – individually and together. CD-ROM includes sample models, image galleries, backgrounds, and textures, plus 3-D software tryout versions ~~$40.00~~ **$36.00**

The Instant Internet Guide by Brent Heslop and David Angell. An Internet jump-start – how to access, use and navigate global networks. 224 pages ~~$14.95~~ **$13.45**

Web Head: The Mac Guide to the World Wide Web by Mary Jane Mara. Published by PeachPit Press. This is a beginning to intermediate book that shows you how to get the most from the Web, using plain talk that beginners will understand, and online veterans will appreciate. There is also instruction on how to build your own home page, posting pages on the Web, and avoiding common HTML mistakes. ~~$24.95~~ **$22.45**

The Underground Guide to Telecommuting by Woody Leonhard, Addison Wesley. There's no place like home. Especially when your boss, your kid, and the neighbor's dog are all barking for your attention simultaneously. Working away from a corporate office presents great (often unexpected) challenges and offers even greater rewards. Woody Leonhard takes on the toughest aspects of telecommuting and gives you the straight scoop on how to make it work for you. Whether you're a telecommuter, a telecommuter's boss, or just curious, The Underground Guide to Telecommuting will give you the tools and information you need to turn electricity and a phone line into major productivity. ~~$24.95~~ **$22.45**

The Elements of E-Mail Style by Brent Heslop and David Angell. Learn the rules of the road in the e-mail age. 208 pages. ~~$14.95~~ **$13.45**

E-Mail Essentials by Ed Tittel & Margaret Robbins is a hands-on guide to the basics of e-mail, the ubiquitous

BOOKS ETC.

networks communication system. The book is suitable for both the casual e-mailer and the networking professional, as it covers everything from the installation of e-mail to the maintenance and management of e-mail hubs and message servers. 250 pp. ~~$24.95~~ **$22.45**

HOT ITEM! **The Computer Privacy Handbook** is a practical guide to e-mail encryption, data protection, and PGP privacy software. With millions of e-mail messages and on-line discussions exchanged daily on the Internet, electronic security has become a key concern. The Computer Privacy Handbook explains practical steps individuals can take to safeguard their electronic security. ~~$24.95~~ **$22.45**

PowerPC System Architecture by MindShare. This book describes the hardware architecture of PowerPC systems, providing a clear, concise explanation of the PowerPC specification, the template upon which all PowerPC processors are designed. The author provides a complete description of the specification for both the 32- and 64-bit implementations. 656 pages ~~$34.95~~ **$31.46**

PCI System Architecture, Third Edition by MindShare. NEW! Describing revision 2.1 of the Peripheral Component Interconnect (PCI) bus specification, this book explores PCI's relationship to the rest of the system. It includes an in-depth treatment of PCI to PCI bridges, the PCI BIOS, the 66MHz PCI bus, and more. 592 pages. ~~$34.95~~ **$31.46**

Cyberpunk Handbook, The Real Cyberpunk Fakebook by St. Jude, R.U.Sirius, and Bart Nagel. Published by Random House. This book tells how to tell if *you* or *someone* you know is a Cyberpunk. ~~$9.95~~ **$8.95**

Planning and Managing Web Sites on the Macintosh by Weiderspan and Shotten. This book is a definitive guide to setting up andrunning a Web site on the Macintosh, written by two experts in the feild. It skillfully teaches you everything you need to know about using WebSTAR, the best known HTTP server software and its shareware predecessor MacHTTP, as well as about writting CGI applications for your server. A special version of WebSTAR, plus tons of useful software, are on the CD-ROM. $35.96

Sex, Lies and Video Games by Bill Hensler is written NEW! for the wannabe games writer locked inside every Mac programmer. This book provides a learn-by-example tutorial on the ins and outs of Mac arcade-style game programming in C. It teaches game theory, sprite animation, sound, and interaction techniques. This book is a must-read for serious programmer's and hobbyists alike. $31.46

Tog on Software Design by Bruce Tognazzini. Respected industry futurist, Bruce "Tog" Tognazzini, presents NEW! his vision of our technological future, detailing the steps computer professionals need to take to deliver new technologies that will profit the industry and benefit society in general. This book contains Tog's insights on a wide range of topics from quality management to the meaning of standards, and responses to queries supplied by designers and developers. $26.96

Foundations™ of Mac® Programming by Dan Parks Sydow. NEW! This all-inclusive tutorial plus reference explains the fundamentals of Mac programming, from resources and memory management to including sound and QuickTime movies. On the CD: reusable Example code for Symante C++ 8.0 or later and Metrowerks Code Warrior compilers, plus shareware and public domain goodies and a searchable form of the book itself. 708 pages, plus one CD-ROM. $39.99

THE APPLE LIBRARY

Learn C on Late Night With MacHack covers the NEW! MacHack conferences from their inception in 1986, up to 1993. Doug Houseman is the program Chairperson of MacHack, and the author of this book. The accompanying CD contains over 100 of the best hacks written at MacHack over the years, including The Grouch, NetBunny, Jurassicon Park, DropSave, QuickTime Balloon Help, the Mac Clapper, Wavy, and more... ~~$29.95~~ **$26.95**

Advanced Color Imaging On the Mac OS explains how NEW! you can augment the color support supplied with Quickdraw, and QuickdrawGX, using the Palette manager to provide the best set of colors on displays with limited color capabilities, soliciting the color choices from users with the color Picker Manager/ Using the ColorSync manager to match colors between screens and input and output devices such as scanners and printers/ learning how the color Manager assists Color QuickDraw in mapping an applications color requests to the actual colors available. ~~$36.95~~ **$33.25**

3D Graphics Programming Using QuickDraw 3D NEW! by Apple Computer, Inc. Now you can incorporate spectacular 3D graphics into your applications. This book/CD-ROM package explores QuickDraw 3D, a graphics extension to the Mac OS for Power Macintoshes. The CD contains the complete QuickDraw 3D system itself and a complete database of the QuickDraw 3D API, allowing you instant access to the hundreds of graphics calls via a fast viewing engine. Book/CD-ROM, 640 pages. ~~$39.95~~ **$35.96**

Apple Guide Complete by NEW! Apple Computer, Inc. For those who want the full power of Apple's complete toolset, this book and CD-ROM package from Apple provides everything you need to produce guide files successfully, including Guide Maker, the software you use to build and test guide files. You'll learn about the complete cycle of designing as well as advanced topics such as scripting and coding guide files. Book/CD-ROM, 544 pages. ~~$39.95~~ **$35.96**

Inside AppleTalk by Gursharan S. Sidhu, Richard F. Andrews and Alan B. Oppenheimer. Apple Computer, Inc. 650 pages. ~~$34.95~~ **$31.45**

AppleScript Finder Guide, English Dialect, by HOT ITEM! Apple Computer, Inc. The AppleScript Finder Guide is an essential reference for anyone who wants to use AppleScript on a Mac to modify existing Finder scripts or to write new ones. The Finder scripting software allows you to write, record, or run scripts that trigger the same desktop actions that you trigger using the keyboard and mouse- actions such as opening and closing folders or manipulating files. This book introduces Finder scripting and describes how to record and modify simple scripts. In particular, it provides definitions for Finder object classes and commands. Use of this book requires that AppleScript be installed, and you should be familiar with AppleScript Scripting Additions Guide, and AppleScript Language Guide. ~~$19.95~~ **$17.95**

HOT ITEM! **AppleScript Language Guide,** by Apple Computer, Inc. The AppleScript Language Guide is the definitive description of the English dialect of the AppleScript scripting language. This book is an essential reference for anyone using AppleScript to modify existing scripts or to write new ones. It also contains useful information for programmers who are working on scriptable applications or complex scripts. This book begins with an introduction to scripting and an overview of AppleScript's main features. Most of the book consists of detailed definitions of AppleScript terminology and syntax in the following categories: Value classes, commands, objects and references to objects, expressions, control statements, handlers, and script objects. In addition to definitions the book provides many sample scripts and discusses advanced topics such as writing command handlers for script applications, the scope of script variables and properties declared at different levels in a script, and inheritance and delegation among script objects. To get the most out of this book, you only need to be familiar with Macintosh computers. Although not required some previous experience with another scripting language (such as HyperTalk) is also helpful. ~~$29.95~~ **$26.95**

AppleScript Scripting Additions Guide, by HOT ITEM! Apple Computer, Inc. AppleScript Scripting Additions Guide is the definitive description of the scripting additions that accompany the English dialect of the AppleScript scripting language. Scripting additions are files that extend AppleScript's capabilities by providing the additional commands or coercions for use in scripts. This book is an essential reference for anyone using AppleScript to modify existing scripts or write new ones. It also contains information for programmers who want to write scripting additions. The Scripting Additions Guide is also a how to install any scripting additions and invoke their commands, to use the standard scripting additions commands, or to write scripting additions. ~~$18.95~~ **$17.05.**

HyperCard Stack Design Guidelines by Apple Computer, Inc. is an essential book for everyone who creates Apple HyperCard stacks, from beginners to commercial developers. It covers the basic principles of design that, when incorporated, make HyperCard stacks effective and usable. Topics include guidelines, navigation, graphic design and screen illustration, text in stacks, music and sound, a sample stack development scenario, collaborative development, and the Stack Design Checklist. 240 pages. ~~$21.95~~ **$19.95**

THE APPLE LIBRARY

Macintosh Programmer's Toolbox Assistant CD-ROM: See page 77

Inside Macintosh®: CD-ROM: See page 78

Inside Macintosh®: Overview by Apple Computer, Inc. is the first book that people who are unfamiliar with Macintosh programming should read. It gives an overview of Macintosh programming fundamentals and a road map to the New Inside Macintosh library. Inside Macintosh: Overview also covers various programming tools and languages, compatibility guidelines and an overview of considerations for worldwide development. 176 pages. ~~$22.95~~ **$20.65**

Inside Macintosh®: Files by Apple Computer, Inc. describes the parts of the operating system that allow you to manage files. It shows how your application can handle the commands typically found in a File menu. It also provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs. ~~$29.95~~ **$26.95**

Inside Macintosh®: Operating System Utilities by Apple Computer, Inc. describes parts of the Macintosh Operating System that allow you to manage various low-level aspects of the operating system. Everyone who programs the Macintosh should read this book! It will show you in detail how to get information about the operating system, manage operating system queues, handle dates and times, control the settings of the parameter RAM, manipulate the trap dispatch table, and receive and respond to low-level system errors. ~~$26.05~~ **$23.45**

Inside Macintosh®: Processes by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to control the execution of processes and interrupt tasks. It shows in detail how you can use the Process Manager to get information about processes loaded in memory. It is also a reference for the Vertical Retrace, Time, Notification, Deferred Task, and Shutdown Managers. 208 pages. ~~$22.95~~ **$20.65**

Inside Macintosh®: Memory by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to manage memory. It provides detailed strategies for allocating and releasing memory, avoiding low-memory situations, reference to the Memory Manager, the Virtual Memory Manager, and memory-related utilities. 296 pages. ~~$24.95~~ **$22.45**

Inside Macintosh®: AOCE Application Interfaces by Apple Computer, Inc. shows how your application can take advantage of the system software features provided by PowerTalk system software and the PowerShare collaboration servers. Nearly every Macintosh application program can benefit from the addition of some of these features. This book shows how you can add electronic mail capabilities to your application, write a messaging application or agent, store information in and retrieve information from PowerShare and other AOCE catalogs, add catalog-browsing and find-in-catalog capabilities to your application, write templates that extend the Finder's ability to display information in PowerShare and other AOCE catalogs, add digital signatures to files or to any portion of a document, and establish an authenticated messaging connection. ~~$40.45~~ **$36.40**

Inside Macintosh®: AOCE Service Access Modules by Apple Computer, Inc. describes how to write a software module that gives users and PowerTalk-enabled applications access to a new or existing mail and messaging service or catalog service. This book shows how to write a catalog service access module (CSAM), a messaging service access module (MSAM), and AOCE templates that allow a user to set up a CSAM or MSAM and add addresses to mail and messages. ~~$26.95~~ **$24.25**

Inside Macintosh®: Devices by Apple Computer, Inc. describes how to write software that interacts with built-in and peripheral hardware devices. With this book, you'll learn how to write and install your own device drivers, desk accessories, and Chooser extensions; communicate with device drivers using the Device Manager; access expansion cards using the Slot Manager; control SCSI devices using SCSI Manager 4.3 or the original SCSI Manager; communicate directly with Apple Desktop Bus devices; interact with the Power Manager in battery-powered Macintosh computers; and communicate with serial devices using the Serial Driver. ~~$29.95~~ **$26.95**

Inside Macintosh®: Macintosh Toolbox Essentials by Apple Computer, Inc. covers the heart of the Macintosh. The toolbox enables programmers to create applications consistent with the Macintosh "look and feel". This book describes Toolbox routines and shows how to implement essential user interface elements, such as menus, windows, scroll bars, icons and dialog boxes. 880 pages. ~~$34.95~~ **$31.45**

Inside Macintosh®: More Macintosh Toolbox by Apple Computer, Inc. covers other Macintosh features such as how to support copy and paste, provide Balloon Help, play and record sound and create control panels are covered in this volume. The managers discussed include Help, List, Resource, Scrap and Sound. ~~$34.65~~ **$31.45**

Inside Macintosh®: Networking by Apple Computer, Inc. describes how to write software that uses AppleTalk networking protocols. It describes the components and organization of AppleTalk and how to select an AppleTalk protocol. It provides the complete application interfaces to all AppleTalk protocols, including ATP (AppleTalk Transaction Protocol), DDP (Datagram Delivery Protocol), and ADSP (AppleTalk Data Stream Protocol), among others. ~~$29.95~~ **$26.95**

Inside Macintosh®: Interapplication Communication by Apple Computer, Inc. shows how applications can work together. How your application can share data, request information or services, allow the user to automate tasks, communicate with remote databases. ~~$34.95~~ **$31.45**

Inside Macintosh®: PowerPC Numerics by Apple Computer, Inc. describes the floating-point numerics environment provided with the first release of PowerPC processor-based Macintosh computers. The numerics environment conforms to the IEEE standard 754 for binary floating-point arithmetic. This book provides a description of that standard and shows how RISC Numerics compiles with it. This book also shows programmers how to create floating-point values and how to perform operations on floating-point values in high-level languages such as C and in PowerPC assembly language. ~~$28.95~~ **$26.00**

Inside Macintosh®: PowerPC System Software by Apple Computer, Inc. describes the new process execution environment and system software services provided with the first version of the system software for Macintosh on PowerPC computers. It contains information you need to know to write applications and other software that can run on the PowerPC. PowerPC System Software shows in detail how to make your software compatible with the new run-time environment provided on PowerPC-based Macintosh computers. It also provides a complete technical reference for the Mixed Mode Manager, the Code Fragment Manager, and the Exception Manager. ~~$24.95~~ **$22.45**

Inside Macintosh®: Sound by Apple Computer, Inc. describes the parts of the Macintosh system software that allow you to manage sounds. It contains information that you need to know to write applications and other software that can record and play back sounds, compress and expand audio data, convert text to speech, and perform other similar operations. ~~$26.95~~ **$24.25**

Inside Macintosh®: Text by Apple Computer, Inc. describes how to perform text handling, from simple character display to multi-language processing. The Font, Script, Text Services, and Dictionary Managers are all covered, in addition to QuickDraw Text, TextEdit, and International and Keyboard Resources. ~~$39.95~~ **$35.95**

Inside Macintosh®: Imaging by Apple Computer, Inc. covers QuickDraw and Color QuickDraw. The book includes general discussions of drawing and working with color. It describes the structures that hold images and image information, and the routines that manipulate them. It also covers the Palette, Color, and Printing Managers, and the Color Picker, Color Matching, and Picture Utilities. ~~$26.95~~ **$24.25**

Inside Macintosh®: QuickDraw™ GX Graphics by Apple Computer, Inc. shows readers how to create and manipulate the fundamental geometric shapes of QuickDraw GX to generate a vast range of graphic entities. It also demonstrates how to work with bitmaps and pictures, and specialized QuickDraw GX graphic shapes. ~~$26.95~~ **$24.25**

Inside Macintosh®: QuickDraw™ GX Objects by Apple Computer, Inc. introduces QuickDraw GX and its object structure, and shows programmers how to manipulate objects in all types of programs. ~~$26.95~~ **$24.25**

Inside Macintosh®: QuickDraw™ GX Environment and Utilities – A companion to QuickDraw™ GX Objects, this book contains programming information useful to any developer writing QuickDraw GX applications. It describes QuickDraw GX memory management, error handling, debugging, and mathematical functions, as well as conversion from QuickDraw to QuickDraw GX. ~~$29.95~~ **$26.95**

Inside Macintosh®: QuickDraw™ GX Library by Apple Computer, Inc. is the powerful new graphics architecture for the Macintosh. Far more than just a revision of QuickDraw, QuickDraw GX is a unified approach to graphics and typography that gives programmers unprecedented flexibility and power in drawing and printing all kinds of shapes, images, and text.

Inside Macintosh®: QuickDraw™ GX Printing This book is essential for any developer whose QuickDraw™ GX application supports printing. It shows how to support the new printing features of QuickDraw GX, including desktop printers and expandable printing dialog boxes. QuickDraw GX Printing also shows how to use printing-related objects to add custom panels to printing dialog boxes and to create custom page formats. ~~$26.95~~ **$24.25**

Inside Macintosh®: QuickDraw™ GX Printing Extensions and Drivers – Any developer who wants to

THE APPLE LIBRARY

create extensions to the application printing capabilities of QuickDraw™ GX, or who needs to write a printing device driver that works with QuickDraw GX needs this book. QuickDraw GX Printing Extensions and Drivers describes how to create printing extensions and printer drivers, and provides a complete reference to the messages, functions, and resources that they use. ~~$29.95~~ **$26.95**

Inside Macintosh®: QuickDraw™ GX Programmer's Overview – This book provides an introduction to QuickDraw™ GX, providing an overview of the QuickDraw GX environment from a developer's perspective. It introduces the QuickDraw™ GX programming and runtime environments, the relationship between QuickDraw GX and the rest of the Macintosh® systems software and the relationship between QuickDraw GX and Macintosh applications. The key elements of QuickDraw GX programming, data structures, object types, and functions used most frequently by QuickDraw GX developers are also covered. After a general introduction, this book provides readers with a series of practical examples demonstrating how to approach programming with QuickDraw GX. ~~$24.95~~ **$22.45**

Inside Macintosh®: QuickDraw™ GX Typography – This book is essential for any developer who uses QuickDraw™ GX to manipulate text. It shows how to use QuickDraw GX objects to handle all kinds of text – from plain, unstyled text to complex, mixed-direction and multi-language text with sophisticated stylistic and typographic variations. QuickDraw GX Typography shows how to create and manipulate the three different types of text shapes supported by QuickDraw GX including text shapes, glyph shapes, and layout shapes. ~~$29.95~~ **$26.95**

Inside Macintosh®: QuickTime by Apple Computer, Inc. is for anyone who wants to create applications that use QuickTime, the system software that allows the integration of video, animation, and sounds into applications. This book describes all of the QuickTime Toolbox utilities. In addition, it provides the information you need to compress and decompress images and image sequences. ~~$29.95~~ **$26.95**

Inside Macintosh®: QuickTime Components by Apple Computer, Inc.covers how to use and develop QuickTime components such as image compressors, movie controllers, sequence grabbers, and video digitizers. ~~$34.95~~ **$31.45**

Inside Macintosh®: X-Ref by Apple Computer, Inc. is a fast access to all the information in Inside Macintosh. Inside Macintosh X Ref; Provides programmers with a quick and easy way to find the exact information they need in this definitive suite of books, (all 26 volumes). It is indexed by topic, volume, chapter, and accompanying page number. ~~$19.95~~ **$17.95.**

Inside the Macintosh Communications ToolBox by Apple Computer. This book is the definitive reference to the Macintosh Communications Toolbox, an integral part of the System 7 Macintosh Toolbox that enables developers to create communications applications or add communications features to other applications.This book describes all of the routines that provide programmers with standard access to important communications services and in addition enables programmers to extend the reach of the Macintosh into non-Apple environments. ~~$24.95~~ **$22.45.**

OpenDoc Programmer's Guide by Apple Computer, Inc. This is the official reference for the implementation of OpenDoc on the Mac OS. The book describes the component software revolution and explains how to develop for it on the Mac OS platform. An accompanying CD-ROM contains a complete reference to the OpenDoc programming interface, and an extensive collection of tested, reusable sample code. **$40.46**

EDITORS/DEVELOPMENT ENVIRONMENTS & LANGUAGES

BBEdit 3.1: See page 77

CMaster 2.0 by Jersey Scientific installs into THINK C 5 / 6 / 7 and Symantec C++ for Macintosh, and enhances the editor. Use its function popup to select a function and CMaster takes you right to it. Other features include multiple clipboards and markers, a Function Prototyper, and a GoBack Menu which can take you back to previous editing contexts. Almost all features bindable to the keyboard, along over a hundred keyboard-only features like "Add New Automatic Variable." Glossaries, AppleScript and ToolServer support, Macros, and External Tools you create too! $129.95

QUED/M 2.7 by Nisus Software, is a programmer's text editor which has defined the industry standard for speed and efficiency. With integrated support for Symantec C/C++, Metrowerks CodeWarrior 6, and MPW, QUED/M offers unrivaled usefulness for the Macintosh developer. In addition to supporting all the major development environments on the Macintosh, QUED/M offers dozens of powerful editing features, including unlimited undo and redo, UNIX style GREP searching, macro language, scripting, text folding, text sorting, file comparison and merging, Toolbox lookup, ten editable/appendable clipboards, line numbering, markers, displaying text as ASCII codes, vertical and horizontal screen splitting, plus much more. $149

CodeManager™ Microsoft® Visual SourceSafe™ 4.0 compatible source code control system for the Macintosh® $399. Call for more info.

SYMANTEC.™

Symantec C++ See page 77

THINK Pascal v. 4.0 by Symantec Corporation. Professionals and students will welcome this version of THINK Pascal. It is fully integrated for rapid turnaround time and lets you take advantage of System 7 capabilities. Features include support for large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking. Product Contents: Four Macintosh disks, a 562-page user manual, and a 498-page object-oriented programming manual. $169

LS Object Pascal CD includes the world's first Object Pascal compiler for Power Macintosh. 100% compatible with Apple's MPW Pascal, LS Object Pascal combines the best of Apple's native development tools with innovative new technology developed at Language Systems. Compiler options specify 68K or native PowerPC code generation. Included on the CD are: LS Object Pascal compiler, Universal Pascal Toolbox interfaces, fully loaded MPW 3.3.1, 68K and PowerPC source debuggers, PowerPC assembler, online documentation, Macintosh Tech Notes, and a special version of AppMaker by Bowers Development that generates native Pascal source code. The beta release includes upgrades to v1.0 when it becomes available. $399

LPA MacProlog comprises a Edinburgh syntax Prolog compiler system set in an attractive multi-window development environment with an integrated program editor, graphical call-graph facilities and an interactive source-level debugger. LPA MacProlog features high-level access to the Macintosh ToolBox for using graphics, dialogs, windows, icons, resources in a simple and versatile way LPA MacProlog also includes interfaces to C and Pascal code resources. The MacProlog Run-time Generator enables the production of double-clickable distributable applica-tions. The compact run-time system supports first argument indexing, tail-recursion and last-call optimization. Optional add-ons tools include flex, Prolog++, MacDBI for Oracle and the MacProlog Dialog Editor. Programmer Edition $745; Developer Edition (which includes the run-time generator and distribution license) $1500

SmalltalkAgents® (STA) is a QKS sophisticated application development environment featuring a new generation of the Smalltalk language, QKS Smalltalk™. Productivity is no longer measured in lines of code, but in project completion time. STA allows "live" direct manipulation of your objects. The development process is dynamic, interactive and iterative. Just like C/C++ and Assembly, STA provides easy and full access to the features of the MacOS™ and Mac Toolbox. You can link your non-Smalltalk code resources using our External Code Linking Toolkit™ (ECLT). You can also call back into STA from foreign functions written in C/C++, Pascal, FORTRAN, and Assembly. STA's automatic garbage collection and object-based typing will free you from tedious memory management and bookkeeping chores. A sophisticated database for source code management provides an almost infinite variety of ways to cross-reference, access, view, and manipulate your code and objects. STA includes an Application Delivery Toolkit™ (ADT) that allows you to create royalty-free, stand-alone, double-clickable applications in just a matter of minutes. The foundation of the QKS Product Family, the Agents Object System (AO/S), is an underlying task framework, housing components and services that save you years of work. Any component built in AO/S will function as an OpenDoc component or container and components from non-AO/S sources can be seamlessly integrated into the AO/S system. SmalltalkAgents List Price: $695.

SOFTWARE ENGINEERING/PROFILERS/DEBUGGERS INSTALLER TOOLS/LIBRARIES/FRAMEWORKS/DATABASES

SOFTWARE ENGINEERING

ICONIX PowerTools by ICONIX has been a leading supplier of CASE tools, since 1984. One of the first Object-Oriented CASE tool developers, ICONIX is known throughout the industry for producing affordable, high-quality tools and state-of-the-art training. Our line of Object-Oriented Analysis and Design tools, ICONIX PowerTools, is an integrated set of 10 CASE tools supporting the major phases of the system development life cycle and automating analysis, design, coding, and the management of complex systems. ICONIX is unique because we are the first to bundle CASE tools, CD-ROM training, and on-site training and consulting. Each individual module is $1,495. PowerPack Bundles: Choose any 6, 8 or 10 distinct modules of your choice to customize your own ICONIX PowerTools toolset. Our professional sales staff will gladly assist you in choosing the right tools for your project's needs. PowerPack/6: $5,995 PowerPack/8: $6,995 PowerPack/10: $7,995. (Full Product Line) Call for pricing on Upgrade Service & Training and Consulting.

Voodoo is a version control tool for the simple and clear management of projects in which files are created in numerous versions (variants and revisions). Voodoo allows both variant and revision control, and it manages not only variants and revisions of single files, but of a whole software project (multi files, multi users, multi variants, access rights, ...). The tool offers a neat graphical user interface and is not only suitable for mere source code control but can handle all different kinds of files with amazing compression rates: typical size of delta between arbitrary files 5% (in words: five per cent) !!!! no matter whether the files are plain text or any other documents – e.g., MSWord, 4D, Canvas, FileMaker ... Please note special prices for multiple copies: single license $190; 2 pack $300; 5 pack $665; 10 pack $1140; 20 pack $2000. Add'l pricing available on request.

PROFILERS/ DEBUGGERS

LJ Profiler by Lars Jordebo Datakonsult supports profiling of C++ 68K and PowerPC applications compiled with CodeWarrior 6, CFront or Symantec C++. Based on active profiling, i.e. profiling code called at function enter and exit, the browser application lets you follow call chain timings in hierarchical views or separate windows. Collect, organize, compare and save profiling data from different versions of your application into a project. Scriptable and recordable with full access to most internal data structures. Optional remote profiling and tracking of segment and stack usage. Full source code to what you link into your application. $295.

Last Resort Programmer's Edition records every keystroke, command key and mouse event (in local coordinates) to a file on your hard disk. This is especially useful for program testing & debugging, and for technical support and help desks. If something goes wrong (because of a power failure, system crash, forgetting to save or deleting lines) and you lose a word, phrase, or document you can look in the Last Resort keystroke file and recover what you typed. Last Resort is also useful for technical support personnel, when they have to ask "What was the last thing you did before...?" $74.95

QC™ by Onyx Technology. See page 69

The Memory Mine™ by Adianta is a stand alone debugging tool for Macintosh and native PowerPC. Programmers can monitor heaps, identify problems such as memory leaks, and stress test applications. Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more... Allocate, Purge, Compact, and Zap memory let users stress test all or part of a program. Source code is not needed to view heaps. It works on Macintoshes with 68020 or later and System 7.0 or later. $99

Spyer by InCider is a simple operated tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed. The recorded data can be saved in files for future use. Spyer works as a background process with any Macintosh application and is triggered by user defined Hot Keys. Spyer enables the "Continuous Redo" utility and is especially useful for software testing and demonstration. $39

INSTALLER TOOLS

InstallerPack™ by StepUp Software is a package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts. Compression formats supported are Compact Pro & Diamond. Each atom also available separately. Compression requires additional licensing. $219

ScriptGen Pro™ by StepUp Software is an Installer script generator which requires no programming or knowledge of Rez. Supports StepUp's InstallerPack, StuffIt compression, custom packages, splash screens, network installs, Rez code output, importing resources, and AppleEvent link w/MPW: $169

LIBRARIES/ FRAMEWORKS/ DATABASES

OOFILE See page 76

PowerTap™ See page 76

3D Game Machine v1.2 by Virtually Unlimited is a C library for creating lightning-fast 3D arcade games and interactive multimedia applications. 3DGM has a simple easy-to-use interface and features very fast rendering (15 frames per second on a 14" monitor completely texture-mapped, with a PowerMac 6100/60), full "virtual" 3D worlds with six degrees of freedom, free-form texture mapping, shading, material and light properties, convex/cave polygons with unlimited vertices, unlimited light sources, dynamic hidden surface removal, special graphic modes for fast full-screen animation, collision detection, explosion simulation, 3D data importing. Runs on all Macs! Works with CodeWarrior. $299 + license.

Animation Class Library version 2.0 (ACL2.0) is an advanced object-oriented multimedia framework, allowing fast development of high-quality interactive applications. Main features of ACL2.0 are: Powerful animation engine which supports structured sprites, collision detection at pixel precision, sprites sorting, powerful blitter and vector objects. Scrolling of background picture in circular buffer and tile-mapscrolling. Application framework for building standard and 3D controls, panes,menus, full screen displays, windows, etc. Quicktime and multi-channel sound support. >800 functions and >100'000 lines. Complete C++ source code for CodeWarrior and Symantec C++, examples, documentation and technical support. $250

dtF is a true relational database system for Apple Macintosh computers. dtF provides a powerful choice for developers who want to create database centered applications with no performance trade-offs. dtF features SQL, full transaction control, error recovery, single user, client server architecture and multi-platform support including DOS, Windows, OS/2 and UNIX. The C/C++ API is identical and fully portable cross all supported platforms. Third-party vendors supporting dtF will be able to offer a variety of advanced features and benefits to their customers royalty free. Tools are included for importing, exporting, creating and managing databases and users. Supported development environments include: Symantec, MPW, Metrowerks and more. Mac/SDK $695

MacWireFrame by Amplified Intelligence. Create your own virtual reality application with MacWireFrame, a virtual reality application frame work. Includes a complete library of object oriented graphics routines, its own easy to understand application frame work (similar to MacApp or TCL but a lot easier to understand), plus an example application program that lets you start solid modeling right away. Comes complete with fully documented source code. All new purchases will be guaranteed a $49.99 upgrade to the soon to be released, scriptable, MacWireFrame 5.0. Due to the overwhelming response the special price offer has been extended for a little while longer. **Special Offer:** ~~$299.00~~ **$75!!!!**

PictureCDEF 1.3 by Paradigm Software is a professional-level CDEF for creating custom graphical buttons (8-64 pixels). PictureCDEF is used in products by Adobe, ProVue, STF Technologies and others. It is multi-monitor and bit-depth sensitive. The button graphic (cicn, ResEdit) can be changed at runtime and even animated with a call-back routine. Create distinct buttons in seven variations: MultiState, PushButton, FlexiButton, ToggleButton, ChkButton, PushPictButton and TogglePictButton. Position the optional button title at left, bottom or right, or follow the system text direction for international support. Manual, sample code and MacApp 3.0 support included. Full source code: $95.00 Object code: $45.00.

Q3S/3dPane/SmartPane source code bundle by Vivistar Consulting. Q3S: source code bundle from ViviStar Consulting. Full featured 3d graphics. Points; lines; polygons; polyhedra; Gouraud shading; z-buffering; culling; depth cueing; parallel, perspective, and stereoscopic projections; performance enhancing "OnlyQD" and "Wireframe" modes; full clipping; pipeline access; animation and model interaction support; and a "triad mouse" to map 2d mouse movement to 3d. **3dPane** provides integration with the TCL and provides a view orientation controller. **SmartPane**

SOFTWARE ENGINEERING/PROFILERS/DEBUGGERS INSTALLER TOOLS/LIBRARIES/FRAMEWORKS/DATABASES

provides TCL offscreen image buffering, flicker free animation, and QuickTime movie recording. SmartPane functions in 3d or 2d scenarios. All work with C++ compilers or ThinkC 6 and compile to PowerPC or 68K target machines. $192

Spellswell 7 1.0.4 is an award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and Fair Witness). Spellswell 7 checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double double word errors, abbreviation errors, mixed case errors, extra spaces between words, a/an before vowel/consonant, etc... MacTech orders include developer kit with Writeswell Jr., a sample Apple Events Word Services word-processor and its source code. $74.95

StoneTable Extra: Additional functions for StoneTable. Drag selected cells within table or to other tables; optionally add rows as part of drag; popup menus or check boxes in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells. Requires StoneTable. (all prices per developer) $50 first compiler, additional compilers $25.

StoneTable: A library replacing all functions found in list manager plus: variable size columns/rows; different font, size, style, forecolor, backcolor per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for Think C, Think Pascal, MPW C, MPW Pascal, CodeWarrior 6 C. (all prices per developer) $150 first compiler, additional compilers $50.

StoneTable and StoneTable-Extra for PowerPC: Same functionality as 68K libraries. Versions for MPW C and CodeWarrior 6 C. Must have 68K libraries. (all prices per developer) StoneTable $100, StoneTableExtra $25.

B-Tree HELPER™ 2.2 is an inexpensive database engine for Macintosh programmers in C source code. B-Tree HELPER gets space in a file in contiguous fixed length blocks. It expands the file as necessary and contracts files when possible. B-Tree HELPER inserts and deletes keys in one or more B-Trees. It finds keys equal to, less than, or greater than a given value in a few hundredths of a second. It finds lists of records whose keys are equal to, less than, or greater than a given value or are in a range of values. $150

AppMaker makes it faster and easier todevelop the user interface for a Macintosh application. Just point and click to design your application, then AppMaker creates resources and generates excellent source code. AppMaker supports most development environments including Metrowerks, Symantec, or MPW; C, C++, or Pascal; procedural or object-oriented, using PowerPlant, TCL, or MacApp. The generated code uses the Universal Headers to provide PowerMac compatibility. Beginners use AppMaker to learn object-oriented and Macintosh Toolbox programming techniques. Experts use it to increase productivity. It saves so much time it's like having your own assistant programmer working for you. Includes one-year subscription on CD. $299

SCRIPTING/SYSTEM ADMINISTRATION

ScriptWizard™ See page 78

FaceSpan™ v2: See page 77

DataScript DataScript is probably he quickest, easiest and most cost-effective way to make your integrated AppleScript solutions database aware-today. **Quickest:** It takes just six lines of AppleScript to make new or existing scripted solutions database aware, and fetch data from RDBM's such as Oracle, Sybase, DB2, or Informix. **Easiest:** DataScript's scripting terminology is easy to learn, easy to use, and easy to remember. "Inside DataScript" contains lots of easy to follow scripts to reuse in your own solution. **Cost Effective:** Because DataScript is so easy to learn, and use you'll become productive very quickly, and once you're ready to ship, you'll find our licensing schemes very attractive. $249.00

Scripter®: See page 77

CLImate by Orchard Software is a command line interface that lets you communicate with your Macintosh using English commands to create, delete, rename, and move files and folders. It can start applications, format disks, restart your computer, etc. CLImate supplements the Finder. It includes a BASIC interpreter that lets you script your Macintosh without AppleScript. The interpreter includes advanced programming constructs: repeat loops, if/then/else conditionals, subroutine calls, etc... CLImate implements wildcard characters, enabling you to work on groups of files. Use CLImate instead of MPW to manage your projects. CLImate is an application occupying 70K disk space. It comes bundled with sample programs and full documentation. $59.95

Cron Manager by Orchard Software implements the UNIX Cron facility. It can open any Macintosh file on a given date and time. By creating an alias, renaming it to the date and time to open, and moving it into the special Cron Events Folder, Cron Manager will open it. Cron Manager is a control panel that creates the special Cron Events Folder inside your System Folder. It is completely transparent to the user. It works like the Startup Items folder, only smarter. It works with any Macintosh file: if you can double-click to start it, Cron Manager can open it. $26.95. Cron Manager bundled with CLImate, $59.95.

Rosanne™ Rosanne is a collection of utilities which offer the user complete control over raw data. Users can sort files, extract selected records, summarize frequency counts, create sample files, perform matching on multiple files, and reformat data to new specifications, all on the desktop, and even on files of a million records or more. The Rosanne Utilities also support AppleScript™, enabling the user to link several actions together to complete an entire process. The Rosanne Utilities are recordable; users may perform a series of actions, and using an AppleScript editor such as Scripter™, see their actions translated directly into AppleScript commands. All of the utilities support multi-tasking and background processing. The Rosanne Utilities will assist you in picking your specifications, determining record length, creating output files and managing the storage of data. **Rosanne Utilities: Copy** – duplicates an input file. **Format** – creates an altered version of an input file, containing either subsets of the fields on the input file, or new fields. **Select** – creates a subset of the records on an input file based on some selection criteria. **The Recode** option allows the user to group data, or correct coding entries. **Sort** – orders an input file by a particular field or set of fields. **Match** – joins together two input files based on common values occurring in corresponding fields or sets of fields. **Aggregate** – creates an output file with summary levels. $595

ScriptBase™ The Scripting Database is a database for storing persistent objects to be made available for access to AppleScript, Apple's system-level user scripting language for controlling applications on Macintosh® computers. Once installed, the database becomes part of the AppleScript system, adding a host of commands to the basic AppleScript vocabulary. Retrieving the objects is simple using AppleScript's natural-language syntax and structure. Objects stored and retrieved in ScriptBase can be accessible any time from any script on the user's computer. These objects can be of any type, including numbers, character strings, lists, records, scripts, and references to disks, files, folders, as well as abstract raw data, to name just a few. ScriptBase can be used to maintain system-wide settings, such as sets of preferences, paths to frequently-used files or folders. Complex installations can be made easier by organizing data and scripts within the database's structure. $79

Script Debugger by Late Night Software Ltd. is a powerful and flexible AppleScript authoring tool. Script Debugger makes it simple for novice and experienced script writers to get the most from AppleScript. The program's advanced debugging environment offers single-step script execution with breakpoints. The Script Debugger dictionary browser features a graphical view of objects provided by scriptable applications. With the program, you also receive the Late Night Software Scripting Additions, a collection of more than 70 new AppleScript commands, and Scheduler, a utility that allows you to launch scripts at pre-determined times. $129

HOT ITEM!

HOT ITEM!

TCP/IP Scripting Addition™ is the latest version of an award-winning AppleScript scripting addition (first place in the 1994 "Best Hack" category in the Everyday AppleScript™ Programming Competition). This scripting addition (or osax) allows you to write scripts using MacTCP™ commands in AppleScript™. Potential uses of this include sending e-mail or files through a script, checking if users are logged on (via Finger), automating FTP, Gopher, NetNews, Telnet, and LPR, verifying links in HTML documents, and quickly writing many other TCP/IP client-server programs. Sample

SCRIPTING/SYSTEM ADMINISTRATION

scripts are included already implementing many of these functions. When combined with FaceSpan, the potential for rapid implementation of Internet client-server applications is enormous. The TCP/IP Scripting Addition works with AppleScript 1.0 or later and MacTCP 2.0.4 or later. It is compatible with Open Transport™. The TCP/IP Scripting Addition can be used from Script Editor, HyperCard 2.2, MacPERL, FaceSpan and other Open Scripting Architecture applications. See "http://www.mangotree.com/biz/mango/index.html" for more details. $49

MACTECH EXCLUSIVES/MISCELLANEOUS

MacTech Magazine is your exclusive source for these specific products including available back issues of SFA's magazine, source code disks and assorted cd's. Call for more info and pricing:

Ad Lib 2.0 The premier MacApp 3.0 compatible ViewEdit replacement. A powerful user-interface editing tool to build views for MacApp 3.0 and 3.1. Ad Lib allows subclassing of all of MacApp's view classes including adorners, behaviors, and drawing environments. String and text style resources are managed automatically. Alternate display methods, such as a view hierarchy window, allow easy examination of complex view structures. Ad Lib includes source code for MacApp extensions that are supported by the editor – buttons can be activated by keystrokes, behaviors can be attached to the application object, and general purpose behaviors can be configured to perform a number of useful functions. Run mode allows the user to try out the views as they will work in an application. Templates can be created to add additional data fields to view classes. Editing palettes provide fast and easy editing of common objects and attributes. Works with ACI's Object Master (version 2.0 and later) to navigate a project's user interface source code. $195

FrameWorks Magazine: $8/back-issue, subject to availability.

FrameWorks Source Code Disk: $10 per back issue, subject to availability.

Five Years of Objects CD-ROM: FrameWorks archives and source code from April 1991 to January 1993, plus selected object-oriented publicly available software and demos. $95

MADACON '93 CD-ROM: The highlights of MADACON '93, including Mike Potel on Pink, Bedrock, MacApp, OODLs, and more. Slides, articles, demos, audio, and QuickTime. $95

MAScript 1.2 adds support for AppleScript to your MacApp 3.0.1 and 3.1 based applications. Make your application scriptable and recordable by building on a tried and tested framework for object model support. MAScript dispatches Apple events to the appropriate objects, creates object specifiers, and makes framework objects like windows and documents scriptable and recordable. Sample application shows you how to begin adding support for scripting and recording. MAScript includes complete source code. Install MAScript by modifying one MacApp source file, then adding another to your project. Future versions of MacApp will incorporate MAScript, so MAScript support you add now will work in the future. $199

The Mjølner BETA System is a software development environment supporting object-oriented programming in the BETA programming language. BETA is uniquely expressive and orthogonal. BETA unifies just about every abstraction mechanism – including class, procedure, function, coroutine, process and exception – into the ultimate abstraction mechanism: the pattern. BETA includes: general block structure, strong typing, whole/part objects. The compiler: binary code generation, automatic garbage collection, separate compilation, interface to C, Pascal, and assembler. The system: persistent objects, basic libraries with containers classes, platform-independent GUI application frameworks on Unix, Mac and Windows NT, metaprogramming system. The tools available on Unix: the hyper structure editor supporting syntax directed editing, browsing, etc., and the source code debugger are currently being ported to the Macintosh system. The Mjølner BETA System for Macintosh requires MPW (basic set) 3.2 or later. Package containing compiler, basic libraries, persistent store, GUI framework, and comprehensive documentation. (Other packages are also available) $295

More Savvy includes all Savvy features plus Apple Event support for all sub-classes of TEventHandler with extensive view support. Apple Event support for text includes text attributes and sub-range specification. Recordability supports additional actions, and coercion includes additional types. Additional client and server Apple Events. $450

Savvy 1.1 OSA support includes attachability, recordability, scriptability, coercion, in addition to script execution, idling and i/o. Apple Event support includes complex object specifiers, synchronous/asynchronous Apple Event handling, and Apple Event transactions for clients and servers. The Core Suite of Apple Event objects is supported including the application, documents, windows, and files. Documentation includes technology overview, cookbook, and sample code. $250 Savvy now supports MPW 3.1, 3.11 and continues to support 3.01, as well as supporting Metrowerks CodeWarrior. **This month only, special offer –** All Savvy versions include free copy of Savvy QuickTime!

NEW Version!

Savvy QuickTime Requires Savvy, More Savvy, or Super Savvy. Includes QuickTime, Apple Event, and view template support. Movies come out of the box ready to play, edit, and react to Apple Events. They can be included in any view structure, including templates, and are displayed in the scrap view. Movie controls include volume, play rate, looping mode, display style, and other characteristics. $250

Super Savvy includes all More Savvy features plus compile, edit, and record scripts using built in script editor. View template editors, like Ad Lib, can attach scripts to view objects and modified scripts are saved with the document. Script action behavior allow quick access for executing and editing scripts attached to views. Text to object specifier coercion plus more. $700

MISCELLANEOUS

REM GOSUB LET NS BASIC

BASIC for the Newton is BASIC for the Newton! From NS BASIC Corporation, it is a fully interactive implementation of the BASIC programming language. It runs entirely on the Newton – no host is required. It includes a full set of functions and data types, hand-written input, windows, buttons and extensions to take advantage of the Newton environment. Applications can create files or access the built-in soups. Applications can also access the serial port for input and output. Work directly on the Newton, or through a connected Mac/PC and keyboard. NS BASIC includes a 240 page pocket sized manual. Runs on all Newton 1.x and 2.0 units. $99

Inside CodeWarrior 8 & **Inside PowerPlant:** See page 101

Guide Composer™ gives anyone the ability to create powerful Apple Guide help systems for any new or existing Macintosh application. Great for commercial developers, shareware developers, in-house developers, and consultants, Guide Composer provides a WYSIWYG development environment: Guide content is developed in Guide windows. Design topics, phrases, and panels in the same format as the user will use them. Features are WYSIWYG interface, Topics, phrases, and hierarchical phrases, Coach marks, Fully-Integrated with Apple's Guide Maker (distributed with Guide Composer), compiles scripts automatically, PICTs in Panels, Generated Guide scripts are modifiable, Compiled files are 100% Apple Guide-compatible and royalty-free. Easy-to-use. $99

MachTen UNIX for Macintosh and Power Macintosh MachTen is a Berkeley UNIX that runs on the Classic to the Power Mac, including PowerBooks and Duos! So, in addition to all of the Macintosh applications, you get a Mach-based UNIX with pre-emptive multi-tasking. MachTen extends the Macintosh operating system with UNIX networking and software development tools. The Macintosh/UNIX integration is so strong that you can even use Macintosh programs and utilities on UNIX data, and UNIX programs and utilities on Macintosh files. Full internet protocol support ensures fast, easy client and server NFS, e-mail, and file transfer between the Macintosh and all TCP-based entities on your network. Built-in internet services include domain name service, POP mail service, internet routing, SLIP & PPP, and Web service. Full X11R5 support with Motif for developing X applications and a high performance X server for using your Mac as an X terminal. MachTen – Power UNIX $695. Personal MachTen (for 68K Macs) $495 . Professional MachTen (for 68K Macs) $695. MachTen X Window Software $350.

Roaster DR1™ See page 100

Geekware by Metrowerks

Mousepad	$8.95
Geekware Hawaiian Classic	$7.95
Blood, Sweat & Code Black Long Sleeve Shirt	$14.95
Cross Platform White Short Sleeve Shirt	$14.95

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

LIST OF PRODUCTS

TIPS & TIDBITS

By Steve Sisak, Contributing Editor

TIP OF THE MONTH

An HGetState Gotcha

`HGetState` does not return a valid handle state when you pass it an empty handle (one whose master pointer is `NULL`). Instead, it returns an error code, so before you call `HGetState`, be sure to check that the handle isn't empty and execute an alternate code path if it is.

I was bitten by this because I was using `HGetState` to determine if a handle was to a resource, and the resource had been purged, so `HGetState` returned an error code instead of the handle flags, and I incorrectly decoded the return result and thought the handle wasn't to a resource.

Eric Schlegel

From Inside Macintosh: Memory, page 1-61 to 1-62:

If an error occurs during an attempt to get the state flags of the specified relocatable block, `HGetState` returns the low-order byte of the result code as its function result. For example, if the handle `h` points to a master pointer whose value is `NIL`, then the signed byte returned by `HGetState` will contain the value -109.

Result codes:

`noErr`	0	
`nilHandleErr`	-109	`NIL` master pointer
`memWZErr`	-111	Attempt to operate on a free block

An Old Bug, Turns Out to be "An Old Bug"

I think I found a bug in a Tip in the March 1995 issue of MacTech magazine. The Tip, entitled "Hot Tip for Hot Keys", can be found on page 67. Allow me to quote a segment of code:

```
else
{
   num = CountDITL(theDialog);
   for(i=0; i<num; i++)
   {
      GetDItem(theDialog, i, &iType, &iHandle, &iRect);

      //and so on ... the omitted code works wonderfully!
   }
}
```

The problem is that the for loop is counting from 0 to Number-Of-DITL-Items minus 1. While the for loop is executed the correct number of times, it's starting and ending one index too early. This off-by-one error is relatively common in C. Either the for loop should be:

```
for (i=1; i<=num; i++)
```

Continued on page 70

*Send us your tips or we'll install EvenBetterBusError on your machine! On the other hand, we might just pay you **$25** for each tip we use, or **$50** for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to our **new Tips e-mail address** at tips@mactech.com. See page two for our other addresses.*

CODEWARRIOR

68K AND POWER MACINTOSH™ Be™OS MAGIC CAP™, WINDOWS 95™, AND WINDOWS NT™/X86

THE MOST VERSATILE PROGRAMMING ENVIRONMENT IN THE WORLD

DISCOVER PROGRAMMING FOR MACINTOSH $79

GOLD 8 $399 BRONZE 8 $149

INSIDE CODEWARRIOR 8 $34.95 INSIDE POWERPLANT™ $34.95

SOURCE CONTROL FOR MACINTOSH: CODEMANAGER DR 1 $399

CodeWarrior®

metrowerks®

MACUSER EDITORS' CHOICE AWARD 1994 WINNER

SOFTWARE PRODUCT OF THE YEAR

To order contact Metrowerks:

voice: (512) 305-0400 fax: (512) 305-0440 e-mail: sales@metrowerks.com

World Wide Web: http://www.metrowerks.com